Fall 2008

Virtual prototyping with surface reconstruction and freeform geometric modeling using level-set method

Weihan Zhang

Department: Mechanical and Aerospace Engineering

VIRTUAL PROTOTYPING WITH SURFACE RECONSTRUCTION AND FREEFORM GEOMETRIC MODELING USING LEVEL-SET METHOD

by

WEIHAN ZHANG

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

IN MECHANICAL ENGINEERING

2008

Approved by

Ming C. Leu, Advisor
Frank W. Liou
Michael G. Hilgers
Robert G. Landers
Robert B. Stone

PUBLICATION DISSERTATION

This dissertation includes the following four articles:

Pages 50-72 have been published in ASME JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING, 2007, 7(3), pp. 203-210.

Pages 73-106 have been accepted for publication in ASME JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING.

Pages 107-119 have been accepted for publication in CIRP ANNUALS - MANUFACTURING TECHNOLOGY.

Pages 120-142 are prepared for submission to the INTERNATIONAL JOURNAL OF COMPUTER AIDED-DESIGN.

ABSTRACT

More and more products with complex geometries are being designed and manufactured by computer aided design (CAD) and rapid prototyping (RP) technologies. Freeform surface is a geometrical feature widely used in modern products like car bodies, airfoils and turbine blades as well as in aesthetic artifacts. How to efficiently design and generate digital prototypes with freeform surfaces is an important issue in CAD. This paper presents the development of a Virtual Sculpting system and addresses the issues of surface reconstruction from dexel data structures and freeform geometric modeling using the level-set method from distance field structure. Our virtual sculpting method is based on the metaphor of carving a solid block into a 3D freeform object using a 3D haptic input device integrated with the computer visualization. This dissertation presents the result of this study and consists primarily of four papers. The first paper presents the development of a novel contour generation algorithm for the purpose of visualizing the sculpted dexel models and interfacing with other CAD/CAM/CAE systems. To improve the sampling quality of the dexel model used in the virtual sculpting system, the second paper develops a triple-dexel structure and a novel surface reconstruction method from triple-dexel data. The developed surface reconstruction method is faster than the voxel-based method, and the reconstructed surface model is more accurate than surface reconstructed from voxel representation using the marching cube algorithm. To enhance the modeling capability of the virtual sculpting system, additional freeform modeling operations including deformation, smoothing and imprint are developed using the user's gesture inputs based on the level-set method. The developed operations generate a water-tight mesh model effective for freeform geometric modeling.

ACKNOWLEDGMENTS

I would like to express my sincere respect and gratitude to my research advisor, Professor Ming C. Leu, for his guidance and support through my academic study and research. During the research work for this dissertation, Dr. Leu shared his knowledge in many areas, provided inspiration and was ready to help whenever I needed his advice. I am fortunate to have such a kind, knowledgeable and passionate advisor in my academic life.

My thanks also go to Professors Frank W. Liou, Michael G. Hilgers, Robert G. Landers, Robert B. Stone and Shun Takai for their valuable suggestions as members of my dissertation committee.

I am also grateful to all the people from the Virtual Reality and Rapid Prototyping Laboratory, whom I had the pleasure to work with. I thank my colleagues, especially, Xiaobo Peng, for his input to my work, for the collaborating in the Virtual Sculpting Project.

I thank my parents for their understanding, confidence, and encouragement while I am concentrating in my research work.

Finally, I would like to thank my girlfriend, Chengchun Tsai, for her endless love and support, limitless patience, and for her efforts and endurance for my study.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

1.1. MOTIVATION

Choosing a product concept is a critical decision-making step in product development. A firm earns a large profit and establishes a strong brand name if it generates and chooses a superior product concept; however, if the situation is reversed, it suffers a huge loss. SONY's Walkman and Betamax video tape are classic examples of such opposite cases [Nathan, 1999]. Because marketplaces have become more competitive and product lifecycles have continued to shorten, firms need to generate and select optimal product concepts more accurately, more inexpensively, and much faster.

In the concept generation phase, once concept variants are computed, virtual prototypes represent a promising alternative to physical prototypes for selecting a final design concept to fabricate. Virtual prototyping can significantly reduce prototyping cost and shorten time taken to evolve a product concept. Virtual prototypes are particularly beneficial to the firms that have never developed similar products before, since the firms can avoid expensive costs investing in tooling and in developing manufacturing processes for physical prototypes.

Commercially available Computer Aided Design (CAD) systems have been widely used for improving the efficiency of the present virtual prototype design process and for better integration with manufacturing systems. However, these CAD systems may not allow the users to implement their ideas on designing virtual prototypes in an intuitive and user-friendly manner. Many traditional CAD systems such as Unigraphics NX, Ideas, etc. have powerful features, but it is not easy to learn and use these tools. Their user interface generally consists of many windows, menus, icons, etc. which tend to bog down the user from concentrating on his/her design intent. Another restriction of the conventional CAD system is in the input devices. The designers use a two-dimensional (2D) input device such as a mouse for the construction of three-dimensional (3D) objects. This restriction causes the use of the modeling system unfriendly and counter-intuitive. Furthermore, in virtual prototyping, the exact dimensions of virtual prototypes are not of main concern. The designer is more interested in creating different part shapes, design configurations, etc. and choosing the most appropriate ones from them. Therefore,

traditional CAD systems, which require precise data for designing objects, are not the best choice for conceptual design.

Virtual sculpting is a process in which the user creates a three-dimensional (3D) object on the computer screen by interactively carving a workpiece like a real sculptor would do on a piece of clay, wax or wood. It is well suited to a freeform design of virtual prototypes as it allows the user to avoid cumbersome interface with the computer [Maiteh et al., 2000; Leu et al., 2001]. Incorporating a haptic interface to the sculpting system also provides the user with a realistic sculpting experience in the virtual prototyping process. Previous researchers [Maiteh et al., 2000; Peng et al., 2006] in Dr. Ming Leu's research labs have developed an experimental virtual sculpting system. The schematic of this system, as shown in Fig. 1.1, is based on the metaphor of carving a solid block into a 3D freeform object. A sculpting tool is controlled by a 3D input device, and the material (workpiece) is represented geometrically by a single directional full-depth pixel data called "single-ray dexel" during the sculpting process. This process starts with a virtual block of material and removes it bit-by-bit by the sculpting tool. Sculpting is implemented via a series of Boolean difference operations that subtract successive tool geometry in a single-ray dexel model from the workpiece. The Boolean difference is performed between the single-ray dexel representation of tool and the single-ray dexel representation of workpiece by comparing the sorted depth data for each pixel. By reducing the complex operations between the tool model and the workpiece model to Boolean operations on one-dimensional segments, the sculpting system can achieve real-time interaction with the human stylist/designer. The PHANToM^{TM} manipulator is used as an input device to provide position and orientation data of the sculpting tool and it is also used as an input device to provide output haptic sensation during sculpting.

However, several limitations of this experimental system have been identified. First, the single-ray dexel data has a low sampling resolution in directions perpendicular to the ray direction such that the generated freeform models may have poor surface representations in those directions. Second, because the single-ray dexel model can be seen only from one direction, the model can only be sculpted from one direction, which greatly limits the modeling capability of the virtual sculpting system. Third, the current system has only limited geometric modeling operations such as material removal and

addition while other intuitive and interactive operations such as deformation, smoothing, and shape copy/pastes have yet been included.

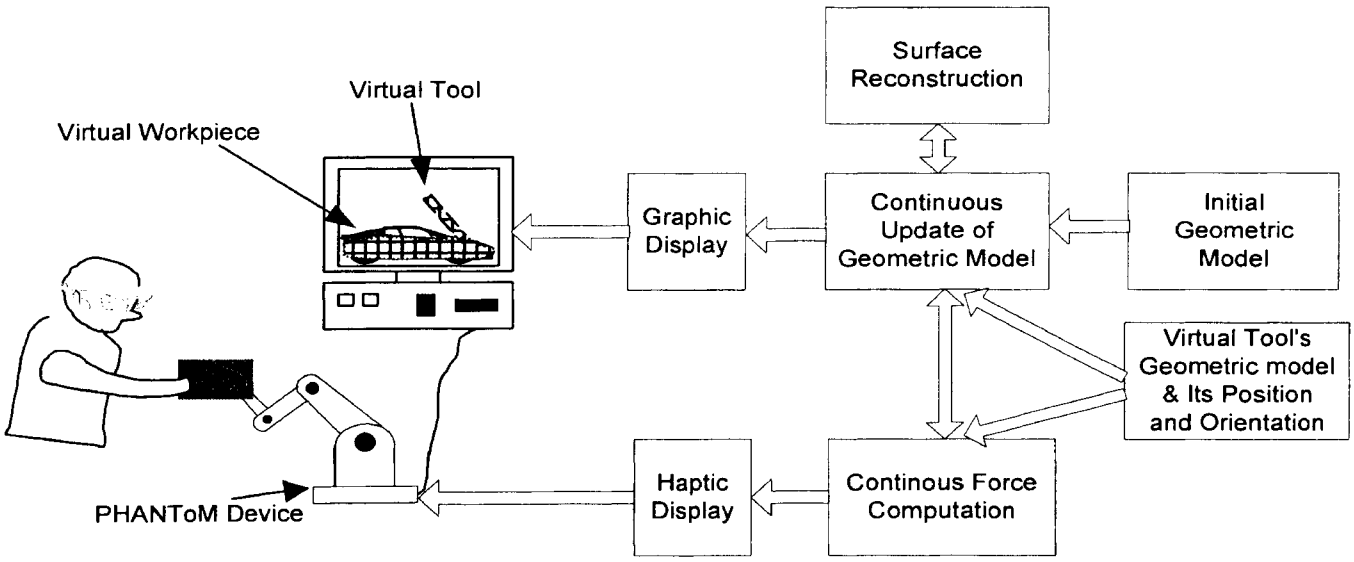


Figure 1.1. Schematic of the System Configuration

1.2. RESEARCH OBJECTIVE AND ISSUES

This dissertation is broadly aimed at improving and extending the freeform geometric modeling capabilities of a virtual sculpting system. A triple-ray dexel based geometric modeling system has been developed to replace the previous single-ray dexel modeling system. A novel surface reconstruction process has been developed to create the surface model from the triple-ray dexel data. The accuracy and computational cost of the developed algorithms have been analyzed. The developed algorithms have been implemented and integrated with the previously developed virtual prototyping system. The interactive level-set method based freeform modeling techniques have been developed and integrated with existing geometric modeling capabilities into a virtual prototyping system that can be used to create 3D concept models of any geometry including freeform surfaces. These techniques will enable intuitive and interactive

generation of any 3D models with haptic interface and subsequent editing of the created models. Material removal, addition, shape deformation and smoothing can be performed using established techniques with the help of the haptic device from SensAble™ Technology [SensAble, 2006].

Developing such a virtual prototyping system is a major undertaking and needs to address many technical challenges in order to meet the stringent requirements on interactive freeform geometric modeling. The fundamental research issues that have been addressed in this dissertation include:

- How to reconstruct triangular surface from the dexel model and triple-dexel model?
- What are the advantages of the developed methods over existing surface reconstruction methods?
- What's the computational complexity and memory cost of the developed algorithms?
- How to formulate the level-set method based freeform modeling framework?
- How to develop fast and efficient freeform modeling operations under this framework?
- How to model the human's gesture inputs and utilize it for freeform modeling?
- How to develop compact data structures to store the geometry information for the level-set method?
- How to develop efficient algorithms for the calculation in the level-set method?

1.3. RELATED WORK

1.3.1. Virtual Prototyping Techniques. Virtual prototyping allows greater communication, productivity and efficiency through realistic modeling and graphic display based on full color, natural texture and appearance. Virtual prototypes are particularly beneficial to the firms that have never developed a similar product before since the firms can avoid expensive costs investing in tooling and in developing manufacturing processes for physical prototypes.

Numerous virtual prototyping techniques have been developed over the years. Zorriassatine et al. [2003] identified five broad classes of virtual prototyping techniques based on the modeling objectives and purposes.

- Visualization

- Fit and interference of mechanical assemblies
- Testing and verification of functions and performance
- Evaluation of manufacturing and assembly operation
- Human factor analysis

Virtual prototyping involves various techniques including geometric modeling, graphics rendering, haptic rendering, etc. Sachs et al. [1991] introduced a system for interactive 3D shape design called "3D draw." Its user interface is based on a pair of Polhemus six-degree-of-freedom tracking devices and a graphic display is used to visualize the scene from a virtual camera position. After drawing and editing 3D curves, which form wire frame models, the next steps are fitting surfaces to groups of linked curves and deforming the surfaces till the required shape is obtained. Deering [1996] at Sun Microsystems created the HoloSketch system, in which the user wears a pair of head-tracked field sequential stereo shutter glasses and manipulates the virtual world through a hand-held 3D mouse/wand. In the design process, the fade-up menu is used to select the required drawing primitives such as rectangular solids, spheres and cylinders or to perform one-shot actions such as cut or paste. Built by researchers at the University of North Carolina [Butterworth et al., 1992] this system uses an HMD to place the designer in a virtual modeling environment. An input device such as a Polhemus 3-space Isotrak held in one hand is used for all interactions including selecting commands from a floating menu, selecting objects, scaling and rotating objects, or grabbing vertices to distort the surface of an object. Researchers at the University of Alberta, Canada developed a system called JDCAD [Liang et al., 1994]. It is a 3D modeling system which uses two 6DOF tracking devices, one to dynamically track the user's head and provide the kinetic 3D effect (e.g. correlation to the position and orientation of head) and the other used as a hand-held "bat" [Ware et al., 1988] to track hand movements. A bat is a tracker that reports 3D position and orientation data. It has three buttons mounted on it for signaling events. By switching modes, the bat can be used to rotate and translate the model under construction to select objects for subsequent operations, and to orient and align individual pieces of the model. Dani and Gadh [1997] presented an approach for creating shape designs in a virtual reality environment called COVIRDS (Conceptual VIRtual Design System). This system uses VR technology to provide a 3D virtual environment in which

the designer can create and modify 3D shapes with an interface based on bimodal voice and hand tracking. A large-screen, projection-based system called the Virtual Design Studio (VDS) is used as an immersive VR-CAD environment. The designer creates three-dimensional shapes by voice commands, hand motions, finger motions, and grasps and shape edits features with his/her hands. Tests have been conducted to compare the efficiency of the COVIRDS with the traditional CAD systems. It was claimed that the COVIRDS system can achieve a productivity of 10-30 times over the conventional CAD systems.

Virtual sculpting, a technique for intuitive virtual prototyping, is "an attempt at the creation of a sculptor's studio-like environment, in which the 'sculptor' can create complex 3-D objects in the computer, as if molding a piece of clay" [Parent, 1977]. This technique is well suited for the development of virtual prototypes because virtual sculpting allows designers to design virtual prototypes in an intuitive and easy manner. Sederberg and Parry [1986] introduced the concept of freeform deformation. Since then, several improvements and extensions have been prompted. The extended freeform deformation method proposed by Coquillart [1990] utilized non-parallelopipedical lattices. Hsu et al. [1992] developed a direct manipulation technique that makes generation and placement of deformations easier. Lamousin and Waggenspack [1994] described a system of NURBS freeform deformations based on a mesh built from rectangular parallelopipeds. Wang and Kaufman [1995] did pioneering research on volume sculpting. Recently Frisken et al. [2000] developed a volumetric sculpting system based on an adaptive distance field, allowing for representation of a volumetric model with adaptive resolution.

Leu and his research team developed a virtual sculpting system to address issues of interactive solid modeling with haptic interface [Maitech et al., 2000; Peng and Leu, 2004; Peng et al., 2005]. The virtual reality interface includes stereo viewing and force feedback. The geometric modeling in this system is based on the Sweep Differential Equation method [Blackmore and Leu, 1992] to compute the boundary of the tool swept volume, and based on the ray-casting method to perform Boolean operations between the tool swept volume and the virtual stock in a single-ray dexel data to simulate the sculpting process. Force feedback is incorporated to enable the user to feel the sculpted

virtual model like actual sculpting with physical materials. Multithreading is used to address the different update rates requirements in graphic and haptic displays.

1.3.2. Geometric Representations. The efficiency and the modeling ability of the virtual prototyping system largely depend on the geometric representation used in the system. Commonly used representations include single-dexel model, triple-dexel model, voxel model, distance field model, and implicit surface model. A review of these representations is given in this section.

1.3.2.1 Single-dexel model. In keeping with the convention on the names pixel and voxel, Van Hook [1986] introduced the notion of single-ray dexel as an abbreviation of "depth element". The single-ray dexel representation of a solid is constructed via computing ray intersections with the solid. For a given solid, parallel and equidistant rays are projected from the viewing direction and intersected with the object as shown in Fig. 1.2.

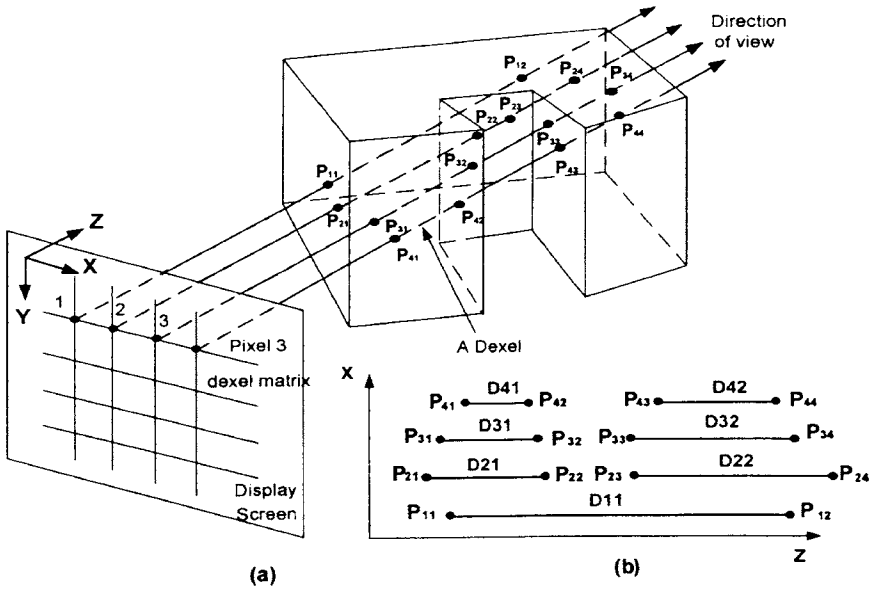


Figure 1.2. The Generation of Dexel Data. (a) The Ray Casting Process and (b) the Single-Dexel Data

For each ray the intersected points with the solid are stored in the following manner: two points defining a line segment that is fully inside the solid make up a dexel. In Fig. 1.2 the two line segments $P_{11}P_{12}$ and $P_{13}P_{14}$ indicate that the points between them are inside the solid. All dexels for a ray are sorted and concatenated into a dexel list and the dexel lists are organized into a dexel matrix. This is the single-ray dexel model.

Using the single-ray dexel data simplifies the implementation of Boolean difference and addition, which compare one-dimensional dexel data between the workpiece and the tool swept volume and manipulate them according to an algorithm for these operations. Taking the single-ray dexel data in the z direction as an example, because the operation is performed on dexel lists at each pixel position, x and y are invariants in the operation, the only variables that have to be considered are (z_{max} , z_{min}) of each dexel. Six relationships between the (z_{max} , z_{min}) of the workpiece and that of the swept volume are summarized in Fig. 1.3. The meanings of the abbreviated symbols in Fig. 1.3 are listed in Table. 1.1 Both Boolean addition and Boolean difference have been implemented and integrated in the virtual sculpting system using single-direction rays.

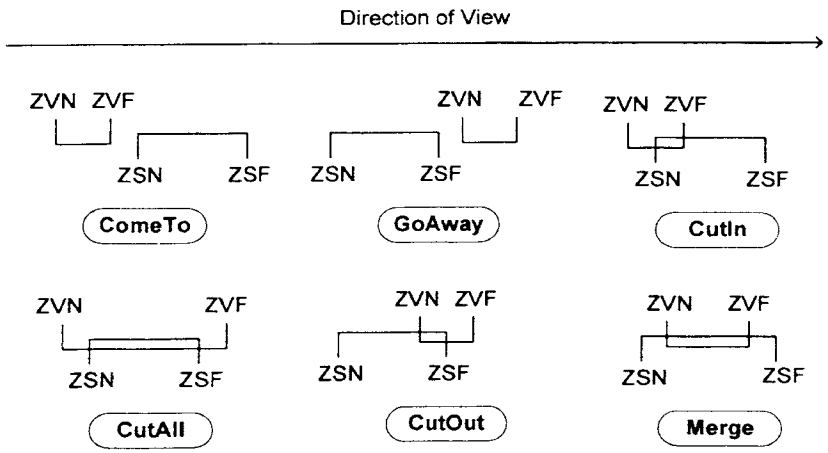


Figure 1.3. Six Possible Relationships between Z_{min} and Z_{max}

Some research has been done previously on the problem of surface reconstruction from single-ray and triple-ray dexel data. Huang and Oliver [1995] briefly described a contour tracking technique to reconstruct contours from a single-ray dexel model without detailed development of an algorithm. The boundary of the object was visualized by simply displaying sets of contours extracted from the single-ray dexel data. Zhu and Lee [2005] presented a visibility sphere marching algorithm for constructing polyhedral surface models from single-ray dexel models for their haptic virtual sculpting. When the algorithm was applied to complex models, some cracks and holes occurred in the generated mesh due to topology related issues [Zhu, 2003]. Benouamer and Michelucci [1997] reconstructed the approximated surface from the triple-ray dexel data by using the marching cube algorithm [Lorensen and Cline, 1987], which still suffers from vast memory cost and ambiguous cases.

Table 1.1. Meaning of Abbreviated Symbols

Symbol	Abbreviation of	Meaning
ZVN	Z Volume Near	The maximum z value of tool swept volume
ZVF	Z Volume Far	The minimum z value of tool swept volume
ZSN	Z Stock Near	The maximum z value of workpiece
ZSF	Z Stock Far	The minimum z value of workpiece

1.3.2.2 Triple-dexel model. In the single-ray dexel model, low sampling quality occurs at surface areas on which the surface normals are perpendicular or nearly perpendicular to the ray direction. To address this problem, an orthogonal triple-ray dexel model has been constructed in this dissertation study by shooting rays in x, y, and z directions, to discretize the model.

The triple-ray dexel model can provide a better resolution than the single-ray dexel model as shown in Fig. 1.4. In comparison with the popular voxel model [Wang and Kaufman, 1995], the triple-ray dexel model requires less computer memory. The memory of the triple-ray dexel model is proportional to the surface area of the geometry, while in the voxel model it is proportion to the volume of the object geometry. If the number of divisions on the x, y, and z axes are N_x, N_y, and N_z, respectively, the memory cost of the voxel model is roughly $O(N_xN_yN_z)$, while the memory cost of the triple-ray dexel model is $O(N_xN_y+N_yN_z+N_zN_x)$. When higher resolutions and more intensive calculations are needed, for example, in the case of adding a distance function onto the voxel model to simulate the interface tracking, the memory-efficient property of the triple-ray dexel becomes an important issue. The triple-ray dexel data is not only memory-efficient but also time-efficient. The access time of the linked list structure used to store the triple-ray dexel data is proportional to the number of elements in the list, where a constant access time can be achieved if a compressed index storage scheme with the knowledge of the connectivity property is used.

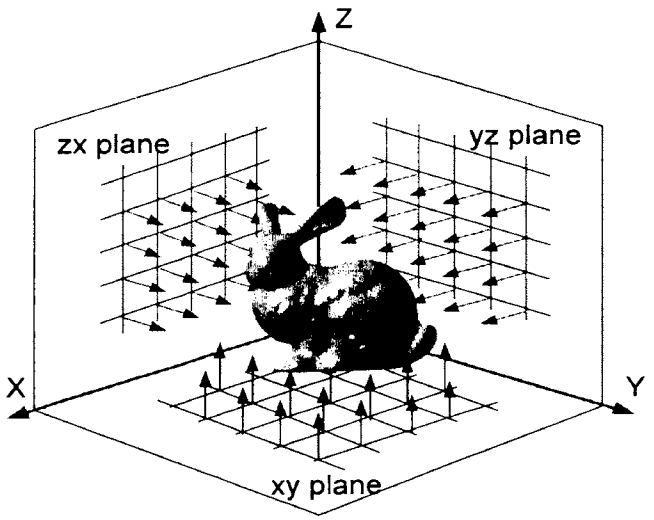


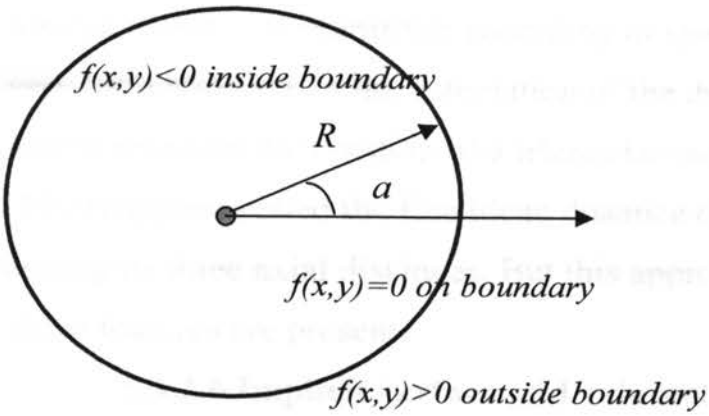
Figure 1.4. Triple Dexel Model

Due to the memory and time efficiency property of the triple-ray dexel data, the triple-ray method has been used by many researchers for different applications. Benouamer and Michelucci [1997] used triple-ray dexel data to convert CSG models into Brep models. Muller et al. [2003] implemented an online sculpting and visualization system by using the triple-ray dexel data. The marching cube algorithm was used to generate the surface from this data. Ren et al. [2006] developed a triple-ray dexel-based virtual prototyping and manufacturing planning system.

1.3.2.3 Voxel model. 3D Geometry can subdivided into small equal-sized cubes. Each cube is called a voxel. A voxel is a volume element, representing a value on a regular grid in three dimensional space. This is analogous to a pixel, which represents 2D image data. Voxel-based representation becomes popular recently in computer graphics, medical image visualization and computer games. The process of converting a geometric representation of a 3D model into a set of voxels is called the voxelization process. There are many freeform operations developed based on the voxel model, such as deformation, smoothing, cutting, addition and etc. A more detailed survey is given by Chen et al. [2001].

1.3.2.4 Implicit model. Implicit surfaces are two-dimensional, geometric shapes that exist in three dimensional space where they are defined according to a particular mathematical function. By definition, if $f(p) = 0$ then p is the point on the surface. f inherently characterizes a volume: those points for which $f < 0$ are on one side (nominally the 'inside') of the surface, those points for which $f > 0$ are on the other side of the same surface. f does not explicitly describe the surface, but implies its existence. For many functions, f is proportional to the distance between p and the surface. A circle in its implicit form is shown in Fig. 1.5.

1.3.2.5 Distance field model. Distance field is a discretized volume representation with distance function, which represents the scalar distances to a surface geometry or shape defined on the vertex of each voxel. It has been used in the computer vision community for image processing, in the physics community for wave-front, Eikonal equation solving, and in the computer graphics community for object representation and processing. Figure 1.6 gives an example of a distance field representation of a 2D contour [Frisken et al., 2000].

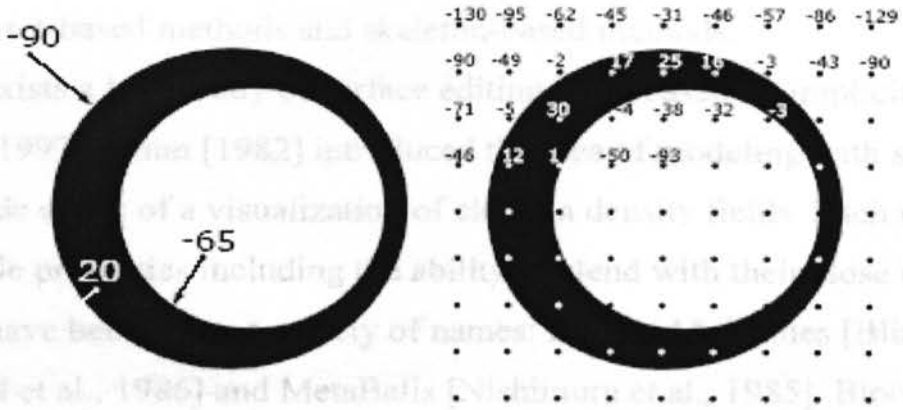


A circle:

- Parametric representation:
$f(x,y)= (R\cos(a), R\sin(a)), a \in [0, 2PI]$

- Implicit representation:
$f(x, y)= x^2+y^2-R^2$

Figure 1.5. A Circle in the Implicit Representation



(a) (b)

Figure 1.6. Sampled Distance Field Data

The implicit surface models can be represented by the discretized distance field data where the distance information is the shortest Euler distance from the grid point to the implicit surface. The available algorithms for computing the distance field from common surface representation includes hierarchical organization and characteristic method [Jones et al., 2006]. Generally a brute force method is used to compute the distances from a grid point in the space to every boundary triangle of M and select the shortest one. To reduce the computation, the shortest distance can be calculated only to a

limited number of primitives according to spatial coherences. However, there has been very little research on the calculation of the distance field directly from triple-dexel data, which precedes the creation of a triangular mesh in virtual sculpting. Sealy and Novins [1999] approximated the Euclidean distance of a grid point as the shortest distance among its three axial distances. But this approximation is not accurate especially where sharp features are present.

1.3.2.6 Implicit surface and volumetric modeling techniques. One of the principal disadvantages of implicit modeling relative to parametric modeling is the difficulty of controlling the shape of an implicit surface [Bloomenthal and Wyvill, 1990] because of the non-intuitive parameters in the implicit function. In order to attack this problem, different direct and indirect implicit surfaces and solid modeling techniques have been developed such as the Blobby models and their extensions, control point-based methods, level-set-based methods and skeleton-based methods.

There exists a large body of surface editing work based on implicit models [Bloomenthal, 1997]. Blinn [1982] introduced the idea of modeling with skeletal implicit surfaces as a side effect of a visualization of electron density fields. Such models have various desirable properties including the ability to blend with their close neighbors. These models have been given a variety of names: Blobby Molecules [Blinn, 1982], Soft Objects [Wyvill et al., 1986] and MetaBalls [Nishimura et al., 1985]. Bloomenthal et al. [1997] pointed out that these models could be grouped under the more general heading of implicit surfaces, defined as the point set $f(r) = 0$ which are called Blobby models. One of the implicit model was given by Blinn [1982] as shown in Fig. 1.7:

$$f_i(x, y, z) = b_i \sum \exp\{-a_i R_i(x, y, z)\} \tag{1}$$

where $R_i(x, y, z) = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2$, $f_i(x, y, z)$ is the field value at any point (x, y, z) created by a primitive P_i at point (x_i, y_i, z_i) and a_i, b_i are variables to adjust the merging of two models. Soft object is another type of the implicit model [Wyvill et al., 1986]. It is developed because the *exp* function is computationally too expensive, thus, it can be approximated by polynomial $f(r)$ as follows:

$$f(r) = a(1 - \frac{4r^6}{9R^6} + \frac{17r^4}{9R^4} - \frac{22r^2}{9R^2}) \quad (2)$$

where $f(0) = 1, f(R) = 0, f'(0) = 0, f'(R) = 0$.

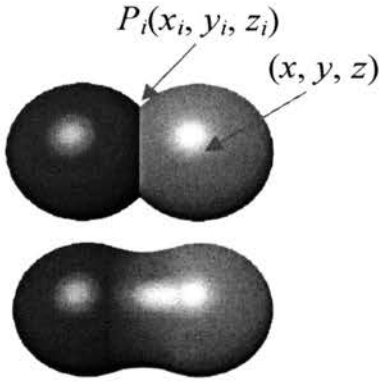


Figure 1.7. The Blobs Model

Given a point in 3D space, the implicit model has the benefit of finding the relation between the point and the surface $f(x, y, z)$. And it is easy to describe the topologically changed surface using, for example, merging, bifurcation, absorbing, etc. Different operations, such as scaling, twisting and CSG operations, are also easy to be defined on a given implicit surface model. Suppose given implicit functions $f(x, y, z) = 0$ and $g(x, y, z) = 0$, according to the definition of the implicit function, different operations on the implicit surfaces such as scale, shear, taper, twist, bend, etc. can be easily defined.

The Blobby models employ local basis functions, so they are often more intuitive to work with than algebraic surfaces [Blinn, 1982]. However, dials or sliders have to be used to adjust the position and radius of each blobby center which is an art work to arrive at the desired surface [Beier, 1993]. Bloomenthal and Wyvill [1990] developed techniques to define/manipulate the skeleton of several shapes, define/adjust the implicit function defined for each skeletal element, and define a blending function to weight the

individual implicit functions. By manipulating the skeletal ellipsoids, the user can produce complex models, and the blending and offsetting operations are controlled by a procedural implicit function which permits a greater degree of localized control as compared to a simple blend of implicit primitives in which each primitive potentially has a global affect on the surface as shown in Fig. 1.8[Wyvill et al., 1999]. But this procedural implicit function is awkward to the end users.

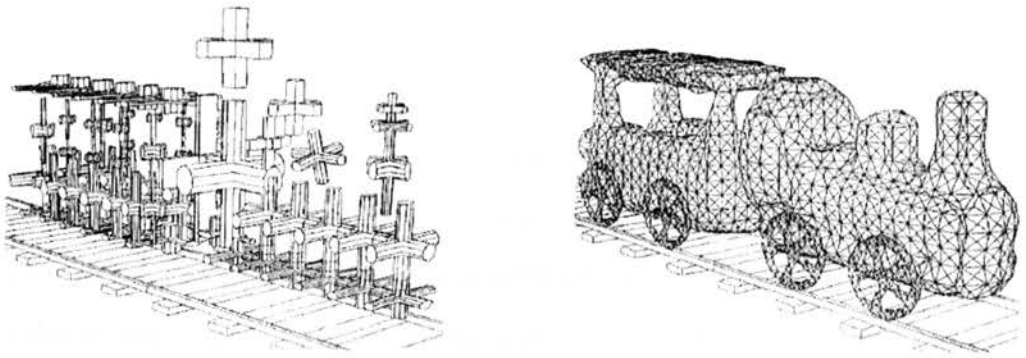


Figure 1.8. Skeletal Elements for the Train and the Surface of the Train after Blending

Wyvill et al. [1999] used tree structure to represent the set theoretical Boolean expression between solid models having half spaces as the primitives. They described techniques for performing blending, warping and Boolean operations on skeletal implicit surfaces called "*Blob Tree*". Galin et al. [1999] addressed the metamorphosis of the Blob-Tree by proposing an original technique that solves the correspondence process and creates an intermediate generic Blob-Tree model whose instances interpolate the initial and final shapes. Besides constructing the model by a larger number of elements arranged in a tree structure, the model can also be defined as single source points with one or several additional curves that control the shape of the field function. Several operations are defined, such as freeform definition curves, rotational, translational or general sweep, twist or interpolation of cross-section, etc. Users can use the splines to control the swept trajectories to generate complex shapes as shown in Fig. 1.9 [Crespin et al., 1996].

Figure 1.9. A Model Defined by Sweeping Primitives

Besides the blobby model based methods, there is another type of implicit function based modeling method called the Control point based methods. Physically based particles provide an interactive means to sample implicit surface functions whereby points on the surface are determined by heuristics, such as the use of the implicit function gradient. Witkin and Heckbert [1994] used particles to sample the implicit surface and applied simple constraint to lock the movement of particles onto a surface while the particles and the surface move. Then, those particles are moved by the user to control the implicit surface. However, it is found that the implicit surfaces are slippery when one attempts to move them using control points. Several approaches [Crossno and Angel, 1997; Rosch et al., 1996] have been proposed to enhance the original Witkin-Heckbert technique by adapting the particle distribution to the local curvature of the surface. Turk and Brien [2002] attacked this problem by using an interpolating implicit surface model and let users directly create and move the boundary constraints to change the shape of the interpolating implicit surface. This provides an intuitive control for interactive sculpting of implicit surfaces which can only accommodate a limited amount of details since at most a few thousand coefficients can be employed in real-time.

Level-set method is a set of numerical methods developed to model the implicit distance field data. Applying level-set methods in interactive geometric modeling is relative new. It started from the work of Museth et al. [2002]. The computational complexity was reduced in their follow-on work [Museth et al., 2005; Nielsen and Museth, 2006]. Museth et al. [2002] developed surface editing techniques like copy, remove and merge level-set models and automatically blend the intersection regions. Their editing operators act on surfaces that happen to have an underlying volumetric

representation, but are based on the mathematics of deforming implicit surfaces. Blending is automatic and is constrained to only occur within a user-specified distance to an arbitrarily complex intersection curve. The user can specify if material should be added and/or removed during editing operations. They also developed a point-attraction operator where a regionally constrained portion of a level-set surface is attracted to a single point. By defining line segments, curves, polygons, patches and 3D objects as densely sampled point sets, the single point attraction operator is combined to produce a more general surface embossing operator. Smoothing and embossing are constrained to occur within a user-specified region, and they are implemented in a level-set framework. They also implement opening and closing morphological operators for performing global blending (closing) and smoothing (opening) on level-set models developed by Sapiro et al., [1993] and Maragos [1996].

Although the same operations such as blending, merging, morphing, Boolean operations, smoothing, embossing, etc. can be implemented by other methods, such as the control point based methods, the level-set method, which uses a simple and physical-based speed function to control the change of the surface, provide a more integrated and intuitive way of modeling the implicit surface. Meanwhile, using level-set methods for modeling guarantees no self-intersection in the generated surface, C^1 continuity in the direction perpendicular to the contour plane, ease of changing topology in freeform shape design, and no edge-connectivity and mesh quality problems associated with mesh models. Given the volumetric representation, the amount of computation time and memory needed to process level-set models is the biggest concern in interactive operations. Additionally, a concern has been raised that volume-based models cannot represent fine or sharp features. Recent advances [Frisken et al., 2000; Kobbelt et al., 2001] have shown that it is possible to model these kinds of structures with volume datasets, without excessively sampling the whole volume. These advances will also be available for the operators once adaptive level-set methods are developed. Comparing with other implicit surface modeling methods, such as the "blob tree" methods, level-set models lack of the skeleton based modeling techniques such as curve skeleton based deformation [Cornea et al., 2005].

In computer graphics, skeletons are widely used for animation [Bloomenthal, 2002; Maya, 2006]. These skeletons (also refer as IK-skeletons) control the polygonal representation of the character being animated. Some shape manipulation techniques are also based on the skeleton methods [Igarashi et al, 1999; Gagvani and Silver, 2001]. The skeleton is defined as the locus of centers of maximal inscribed (open) balls included in the geometry [Lieutier, 2004]. The process of obtaining a skeleton is called skeletonization, which can generate results such as those shown in Fig. 1.10[Cornea et al., 2005]. The line-like skeleton representation of a 3D object is called the curve skeleton [Svensson et al., 2002]. Different implicit geometric modeling techniques have been developed by using the skeleton information of the model. Sederberg and Parry [1986] developed a freeform deformation technique, where an object is enclosed in a parallelepiped and its deformation is defined by using a vector transformation to deform the parallelepiped.

Overall, implicit functions represented surface or volume model is hard to modify due to the use of non-intuitive parameters in the implicit function. Although different techniques have been developed to address this problem, such as the Blobby models, the control point-based methods, etc., the results are not fully satisfied. Based on real physics, level-set methods provide a uniformed framework to model the implicit models and build up the connections between pure geometric modeling and physical laws. However, how to build intuitive user interface and operations to control the speed function in the level-set methods to modify the shape remains an open research question.

1.3.3. Surface Reconstruction for Virtual Prototyping. The conversion from any geometric representation of a 3D model into triangular surface patches is an important issue. It is because the reconstructed triangular facets can be used by conventional CAD/CAM/CAE systems to perform geometric design, engineering analysis, and automated manufacturing applications. Further, the triangulated 3D model can be viewed in any directions as desired using standard routines of computer graphics software. However, the surface reconstruction from discretized geometric representations such as dexel, voxel structures is difficult because reconstruction methods have to overcome topological ambiguity, which is usually being dealt through grid based methods.

Figure 1.10. Examples of Curve Skeletons of Different 3D Objects

1.3.3.1 Surface reconstruction from dexel model. Dexel data is view-dependent because it only records the geometric information of a 3D object from one viewing direction. In the practice of dexel-based NC simulation, researchers were only able to produce a limited number of views from certain directions for the simulation, without the generation of a surface model that can be viewed from any directions. To solve the view-dependent problem, Huang and Oliver [1995] briefly described a contour tracking technique but without detailed development of an algorithm. They visualized the boundary of the object by simply displaying sets of contours extracted from the dexel data. Konig and Groller [1998] described an algorithm to create a surface representation from dexel data for 3-axis milling simulation. But the algorithm could fail easily in the virtual sculpting process where dexel data are modified in arbitrary directions. Zhu and Lee [2005] presented a visibility sphere marching algorithm for constructing polyhedral models from dexel data for their virtual sculpting research. When the algorithm was applied to complex 3D objects, there could be some cracks and holes in the generated mesh due to topology related issues [2003]. The Marching Cube Algorithm [Lorensen and Cline, 1987] has been used to generate an approximate triangular surface from tri-dexel data [Benouamer and Michelucci, 1997] and from voxel data. But this algorithm requires huge memory storage and suffers from some ambiguity, and it can not be applied to dexel data generated in a single direction. Muller et al. [2003] implemented the point-based rendering method developed by Pfister et al. [2000] for their online sculpting system. However, it was difficult to interface the sculpted models with CAD/CAM/CAE systems for further design and analysis.

Another line of related research is the curve reconstruction study in computational geometry stated as follows: given a set of sample points from a curve, a reconstruction of the curve is intended, i.e., points are to be joined by edges in the order they appear on the curve. The dexel points can be seen as the points on the curves in relation to this study. The developed methods included the α-shape [Edelsbrunner et al., 1983], β-skeleton [Kirkpatrick and Radke, 1985], and γ-neighborhood graph [Veltkamp, 1992]. But all of them require certain preconditions on the input points. The α-shape method works well for the points which are evenly distributed in the interior of an object. The β-skeleton method requires the sampling density of points varied with the local feature size on the curve. These curve reconstruction methods can not be directly applied to dexel data due to the nature of their input data.

Another related research is the study of surface reconstruction from point clouds since dexel data can be treated as point cloud data in 3D space. Literature in this research comes mainly from the fields of image processing, computational geometry and computer graphics [Azernikov et al., 2003]. Delaunay-based methods [Edelsbrunner and Mucke, 1994; Bernardini et al., 1999; Amenta et al., 2001; Dey et al., 2001] have been shown successful to produce a triangular mesh from point cloud data. However, the ball-pivoting algorithm [Bernardini et al., 1999] took 2.1 minutes to reconstruct 361K samples on 450MHz Pentium II Xeon PC, and the power crust method [Amenta et al., 2001] took about 6 minutes to reconstruct 30,000 samples on a 400 MHz Sun computer. Besides Delaunay-based methods, surface fitting techniques [Carr et al., 2001; Alexa et al., 2001; Ohtake et al., 2003, 2006] have become popular recently for surface reconstruction because of their ability to account for noise in the input data. Nevertheless, one of the fastest implicit surface fitting methods [Ohtake et al. 2006] still took 42 seconds to reconstruct the surface from a 362K input data on a 1.6 GHz Pentium IV PC.

1.3.3.2 Surface reconstruction from planar contours. Surface reconstruction from a set of planar sectional contours has been an intriguing problem in diverse research areas. This problem arises primarily in the fields of medical imaging, digitization of objects, and geographical information systems. Keppel [1975] described an algorithm for obtaining an optimal approximation, using triangulation, of a three dimensional surface defined by randomly distributed points along contour lines. Fuchs et al. [1977] presented

a general solution by determining an optimal surface between each pair of consecutive contours. Wang and Aggarwal [1986] developed a versatile surface representation which preserves the local object surface structure. Sloan and Painter [1987] described a test bed for evaluating all the known reconstruction techniques and presented an improvement on the simple divide-and-conquer method analyzed by Fuchs, Kedem, and Uselton earlier. Boissonnat [1988] proposed a new solution by constructing a volume whose boundary is a polyhedron with triangular faces intersecting the cutting planes along the given contours. Meyers et al. [1992] developed a method which produces a triangulated mesh from the data points of the contours. The method is then used in conjunction with a piecewise parametric surface-fitting algorithm to produce a reconstructed surface. Oliva [1996] proposed an algorithm which constructs the surface for any non-self-intersecting contours by means of adding an appropriate number of intermediate cross-sections between complicated contours and triangulation of every pair of contours in different slices. Later Felkel and Obdrzalek [1999] proposed a modification of Oliva's method for reconstruction of 3D surfaces from contours in parallel cross-sections. Bajaj et al. [1996] developed a surface-based algorithm which achieves both faster rendering and lower likelihood of reconstruction errors.

Cheng and Dey [1988] improved a Delaunay triangulations based method and it seemed to be more promising and appropriate in handling correspondence and branching problems. Felkel and Janacek [1999] implemented two approaches for reconstruction of 3D objects from contours in serial sections. The first method is based on thresholding and 3D volume reconstruction, the second on direct reconstruction from parallel contours. Treece et al. [2000] proposed a Shape-based interpolation method which is a simple, efficient and fast surface reconstruction technique for contour data-sets. Klein et al. [1999] used the concept of distance field for a robust reconstruction algorithm, which is based on the medial axes.

1.3.3.3 Surface reconstruction from volumetric models. There are many surface rendering algorithms that reconstruct triangular surfaces from the voxel data structure includes marching cube algorithm [Lorensen and Cline, 1987], marching tetrahedrons [Doi and Koide, 1991], marching triangles [Hilton et al., 1996] and etc.

The marching cube algorithm is the most popular one because of its easy implementation and fast computational speed. The algorithm takes eight neighbor locations of an imaginable cube at a time, then determining the polygon(s) needed to represent the part of the isosurface that passes through this cube. This is done by creating an index to a precalculated array of 256 possible polygon configurations ($2^8 = 256$) within the cube, by treating each of the 8 scalar values as a bit in an 8-bit integer. If the scalar's value is higher than the iso-value (i.e., it is inside the surface) then the appropriate bit is set to one, while if it is lower (outside), it is set to zero. The final triangular surface is generated after going through this process for all the cubes.

If the input data is the distance field data, then the distance value on each node can be utilized as the scalar value to determine if this grid is inside or outside the boundary. After applying the same approximation algorithm to find the triangles inside each cube, the triangular surface can be generated from the distance field data.

2. RESEARCH TASKS AND MAIN RESULTS

2.1. SURFACE RECONSTRUCTION FROM DEXEL DATA

The objective of studying the surface reconstruction from both single-dexel and triple-dexel data is to develop a fast surface reconstruction method to reconstruct a triangular surface from dexel structures for the purpose of visualization and interface with other CAD/CAM/CAE systems. The following tasks have been accomplished in this dissertation work.

2.1.1. Contour Reconstruction from Dexel Model. The difficulties of reconstructing planar contours from dexel data arises when there exist inner contours on slices taken from a 3D model with interior voids. Our approach to address the inner-contour difficulty is to design an algorithm that dictates how to connect dexel points on two adjacent rays for any considered planar slice by separating the dexels into groups. This requires the development of a grouping criterion, which categories the dexels on two adjacent rays into different groups. The main idea behind our design of the grouping criterion is the observation that two overlapping dexel spaces on two adjacent rays may form part of an inner contour. An illustration of this observation is given in Fig. 2.1. One slice of the 3D model on XZ plane in Fig. 2.1(a) has dexel data shown in Fig. 2.1(b). The overlapping dexel spaces between points 6 and 7 and between points 12 and 13 form an inner contour because the top of these overlapped dexel spaces is covered by dexel B and the bottom is covered by dexel A.

According to this observation, if a set of overlapping dexel spaces is covered by both a dexel beneath the bottom and a dexel right above the top, these dexel spaces form an inner contour and are called *a closed set*. For example, the set of dexel spaces between points 6 and 7 and between points 12 and 13 in Fig. 2.1(b) is a closed set. The connections of a closed set of dexel spaces to form an inner contour are: filling the top and the bottom dexel spaces, and connecting the boundary dexel points on the same side of the dexel spaces accordingly (e.g. connecting point 6 and point 12, and connecting point 7 and point 13 in Fig. 2.1(b)). Meanwhile, if a set of overlapping dexel spaces is not covered by both a dexel beneath the bottom and a dexel above the top, it is *an open set* of dexel spaces. Their dexel points need to be connected differently to form part of an outer

contour. For example, the dexel spaces between points 4 and 5, between points 10 and 11, and between points 16 and 17 in Fig. 2.1(b) are an open set. The connections of an open set of dexel spaces are: filling the top or the bottom dexel space, depending on which is covered by a dexel above or beneath, and connecting the boundary dexel points on the same side of the dexel spaces accordingly.

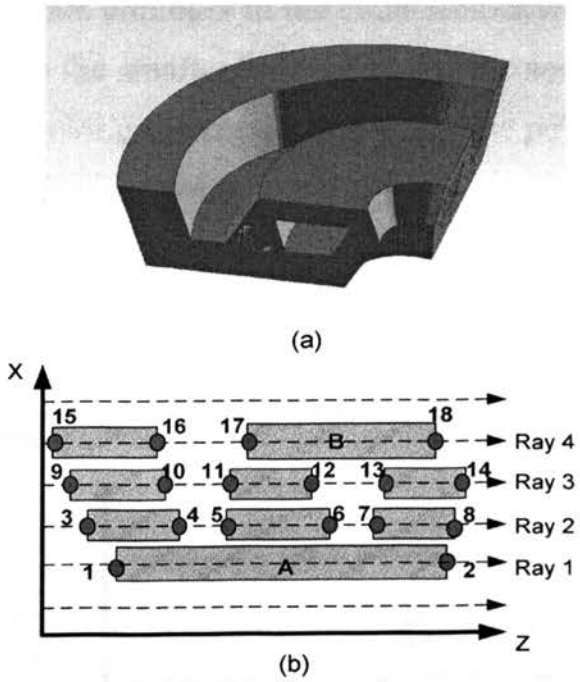


Figure 2.1. Example of the Contour Generation Algorithm. (a) 3D Model (b) One Slice of the 3D Model on XZ Plane

By using the grouping criterion, a four-step contour generation algorithm has been developed. The algorithm first categorizes the dexels on two adjacent rays into different groups by using a "grouping" criterion. The dexel points in the same group are connected using a set of rules to form sub-boundaries. After checking the connections among all the dexel points on one slice, a connection table is created and used to obtain the points of connection in a counterclockwise sequence for every contour. Finally, the contours on all the parallel slices are tiled to obtain triangular facets of the boundary surface of the 3D

object. To illustrate the contour generation process, Figure 2.2 is used as an example. On Ray 4 and Ray 5, dexels $d_{4,2}$, $d_{4,3}$, and $d_{5,1}$ are in one group because they have overlaps, and $d_{4,1}$ is in another group. Thus, points 12 and 13, points 11 and 15, and points 14 and 16 are connected. Because dexel $d_{4,1}$ is a top dexel, points 9 and 10 are connected. Once all the connections are made for every two adjacent rays, a connection table can be created and all the connections are listed in the table as shown on the left side of Fig. 2.2, where the middle column lists the dexel points in the same sequence as they are generated and read. Their connecting points are stored in the left and right columns separately. In order to generate contours in the counterclockwise direction, the left column is always filled with the smaller index. Finally, the sequence of points for each contour is generated by following the connection from one point to the next, until eventually coming back to the first point.

Connected Point	Dexel Point	Connected Point
2	1	3
1	2	8
1	3	9
5	4	10
4	5	11
7	6	12
6	7	13
2	8	14
3	9	10
4	10	9
5	11	15
6	12	13
7	13	12
8	14	16
11	15	16
14	16	15

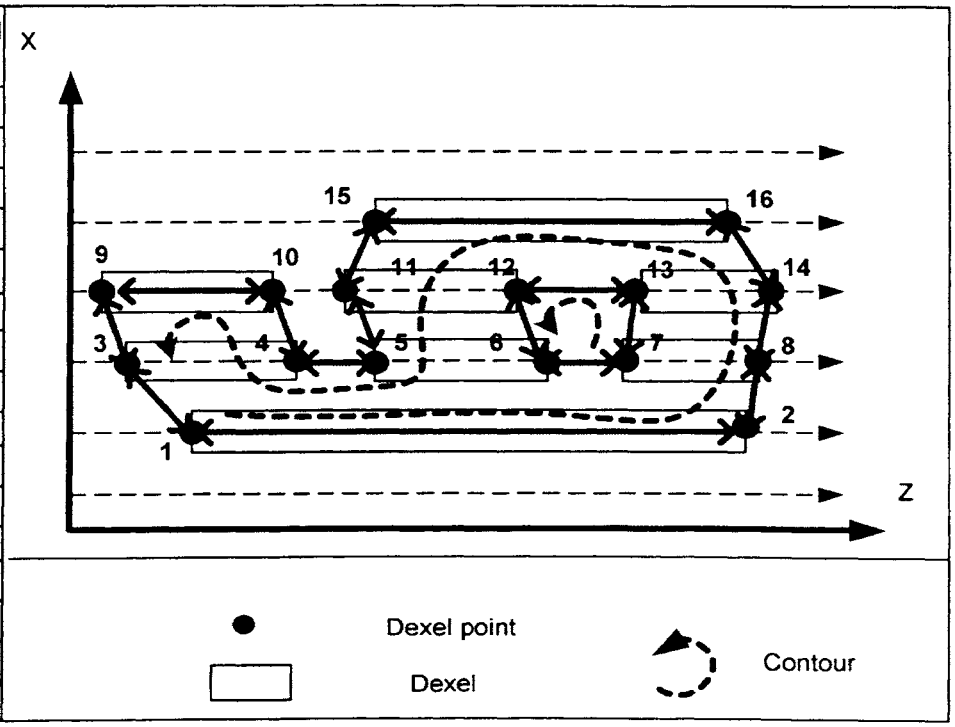


Figure 2.2. Contour Generation from Single-Dexel Data

2.1.2. Surface Reconstruction from Planar Contours. The methods of contour generation and surface reconstruction presented above have been coded into computer software and incorporated in the development of an experimental virtual sculpting system and an NC machining simulation system. In order to finally generate the triangular facets of the object's boundary surface for viewing purpose, the algorithm developed by Christiansen and Sederberg [1978] has been implemented to reconstruct the surface model from planar contours. Figure 2.3 shows a freeform cat model created within the virtual sculpting system. The original cat model (without eyes in Fig. 2.3(a)) is imported into the system in the STL format. Eye cavities are first carved by sculpting the cat model with cylinder shaped cutters. Two eyeballs are then added and placed in the cavities by performing Boolean union with ball shaped cutters. The tail of the cat is also added. After applying the contour generation algorithm and tiling the generated contour into a triangular surface patch, the modified cat model can be viewed in any directions as shown in Fig. 2.3(c) and (d).

2.1.3. Surface Reconstruction from Triple-Dexel Model. The main idea of the proposed surface reconstruction method is to generate contours from triple-dexel data on three sets of orthogonal slices, and utilize these contours to reconstruct the boundary surface of the 3D model. Overall, the method has three main steps. First, the contour generation algorithm takes the dexel data in each of x, y, and z directions as the input and generates planar contours on two orthogonal sets of parallel slices. For example, the dexel data in x direction is used to generate xy contours and xz contours. Next, on each set of parallel slices, the two sets of contours generated from the first step are combined into one set of contours. For example, an xy contour is combined with a yx contour on the same slice to generate a contour parallel to xy plane. After these two steps, there are three sets of contours (i.e., contours on planes parallel to xy, yz and zx planes). In the last step, a volume-based tiling algorithm is utilized to generate triangular facets of the solid's boundary surface from the three sets of contours. The schematic diagram of the proposed method is shown in Fig. 2.4.

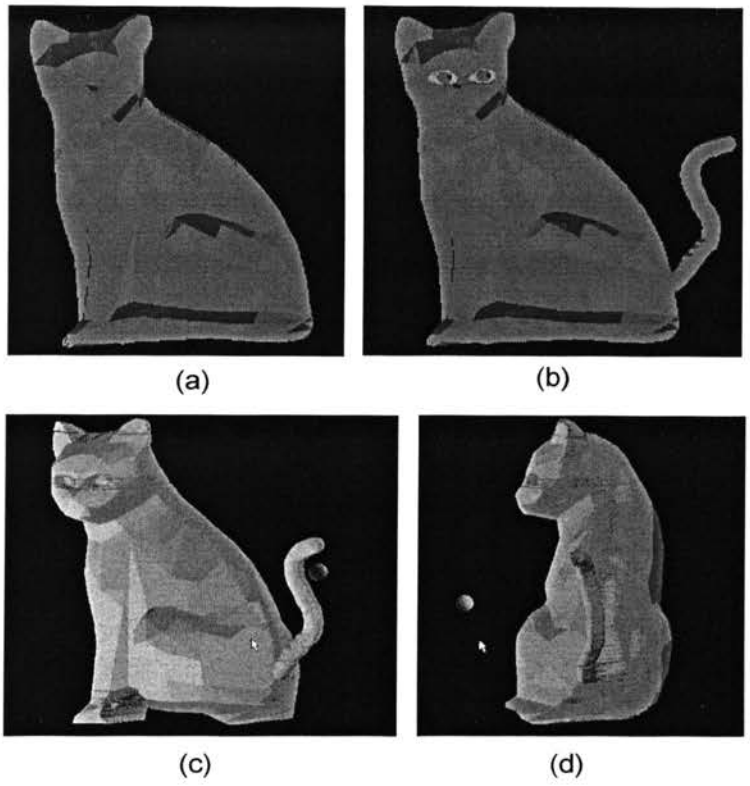


Figure 2.3. Modeling Example of a Cat Model. (a) The Imported Cat Model Created from a CAD System (b) Eyes and Tails Created by Virtual Sculpting (c) and (d) Viewing the Modified Cat Model in Different Directions

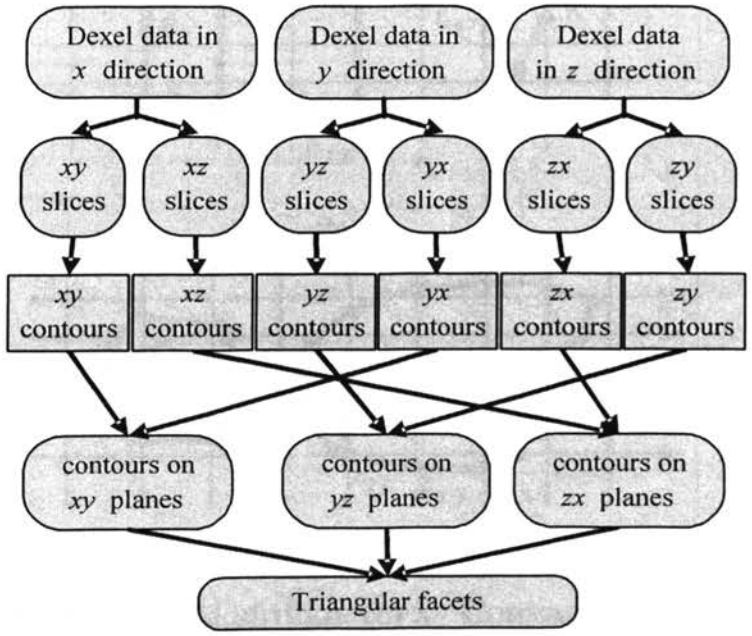


Figure 2.4. Proposed Method of Surface Reconstruction from Triple-Dexel Data

2.1.4. Contour Combination. After applying the contour generation algorithm to triple-dexel data, two sets of contours are present on planes parallel to each of *xy*, *yz* and *zx* planes. The objective of the contour combination algorithm is to correspond and combine these two sets of contours into one set of contours to more accurately represent the cross-sectional profiles of the 3D model. For example, in Fig. 2.5, contour A_1 generated from dexel data in *x* direction is corresponded with contour B_1 generated from dexel data in *y* direction to create contour C_1. Likewise, contour A_2 is corresponded and combined with B_2 to generate contour C_2.

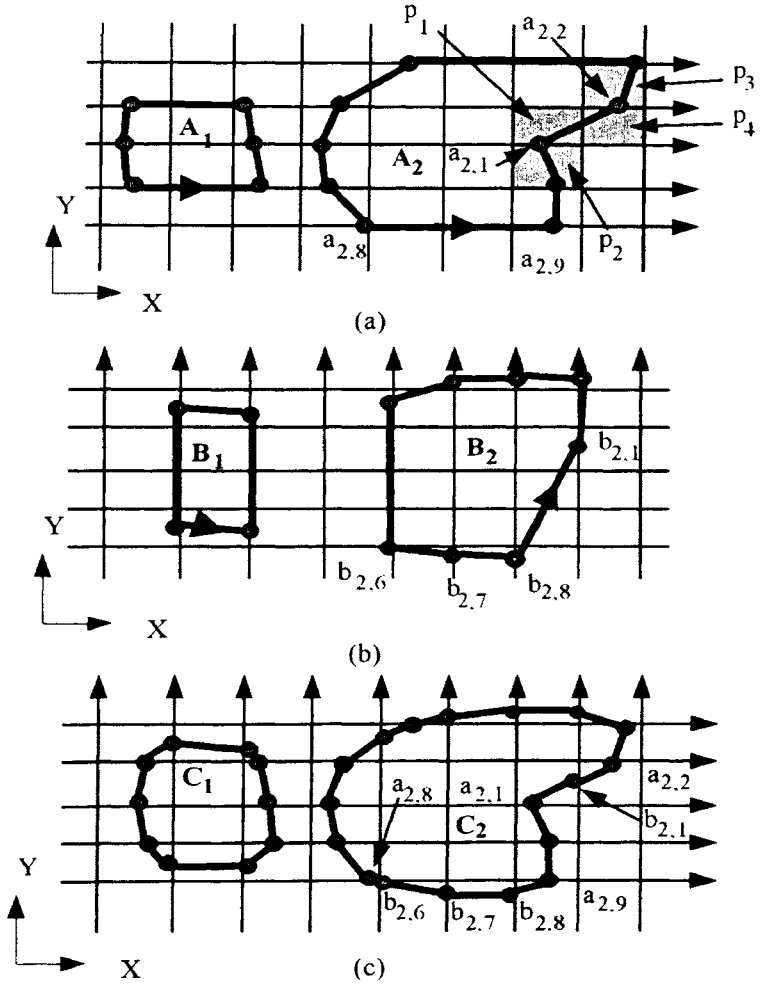


Figure 2.5. Contour Combination Algorithm. (a) xy Contours, (b) yx Contours and (c) the Combined Contours

The contour correspondence problem involves finding which contour from contour set A is to be combined with which contour from contour set B. The overlapping area ratio [Wang and Aggarwal, 1986] between two contours has been utilized as the criterion to deal with this correspondence problem. The overlapping ratios between A_i and all the contours from contour set B are firstly calculated. Then the contour which has the maximum overlapping ratio with A_i is chosen. Likewise, every other contour in contour set A can be corresponded with a contour in contour set B. Here the numbers of input contours from each set is assumed as equal. To speed up the calculation, the overlapping area between contour A_i and contour B_j is approximated by the overlapping area of their bounding boxes.

The contour combination algorithm consists of two main steps to combine the corresponded contours (say, A_i and B_j). The first step is to identify the starting pair of points $a_{i,k}$ and $a_{i,k+1}$ of contour A_i, to find their associated points (i.e., $b_{j,f}, \ldots, b_{j,l}$) and to add them between $a_{i,k}$ and $a_{i,k+1}$. The second step is to continuously search from $a_{i,k+1}$ and $a_{i,k+2}$ to find the next pair of points in contour A_i which has at least one associated point from contour B_j. Then the associated points are identified starting from $b_{j,l+1}$ and onwards in B_j for insertion. The second step is repeated until all the points from contour B_j have been added to contour A_i.

2.1.5. Surface Reconstruction from Three Orthogonal Slices of Contours. After the contour combination process, three sets of orthogonal slices of contours are generated. The volume-based tiling algorithm of Svitak and Skala [2004] is utilized to reconstruct the boundary surface of the 3D model from these contours. The main idea of this volume-based tiling algorithm is to generate triangular facets within each rectangular box associated with the rays in x, y and z directions. Because the three sets of orthogonal contours contain the positions and connectivity of all triangle vertices, the problem of generating triangular meshes within each box becomes the problem of searching the locations and connection information of the vertices from the three sets of contours that have been generated. Once this information is obtained, it is trivial to generate the triangular facets within each box by using a triangular patching algorithm.

The volume-based tiling algorithm consists of three steps. Given a triple-dexel data with M, N, and O numbers of divisions in the x, y and z axes, respectively, the 3D

space is divided into $M \times N \times O$ equal-sized rectangular boxes. The algorithm first identifies the Boundary Sub-Volumes (BSVs) that are the boxes having non-null intersections between their edges and the solid's boundary surface. Second, the three orthogonal sets of contours are searched to find a close loop of vertices within each BSV. Finally, triangular facets are created within each BSV by patching these vertices.

2.1.6. Computational Complexity Analysis. The computational complexity and storage requirement of the contour generation algorithm are analyzed. Two test cases have been utilized to verify the computational complexity analysis. The computational complexity of the contour generation algorithm is $O(\alpha^2\beta)$ for each slice where α is the average number of dexels along a ray and β is the number of rays intersecting with the object for the considered slice. For the triple-dexel model, the total computation complexity for the contour generation algorithm is $O(\alpha T)$ where T is the number of dexel points in the triple-dexel model. The complexity of the contour correspondence and combination algorithms is $O(T)$, where T is the total number of dexel points. The memory costs of the contour generation and contour combination algorithms are linearly proportional to the number of dexel points of the triple-dexel model.

2.1.7. Surface Error Analysis. The reconstructed surface is watertight because in the volume-based surface tiling algorithm, every dexel point inside the boundary sub-volume is guaranteed to have connection points to form a close loop. However, the reconstructed surface is still an approximation of the original shape. To estimate the quality of the reconstructed surface, the reconstructed surface error is defined as the ratio of the Hausdorff distance between the original surface and the reconstructed surface to the diagonal length of the bounding cuboid. The surface errors of the reconstructed Stanford bunny model from triple-dexel data are calculated using the Metro [Cignoni et al., 1998] comparison tool under four different resolutions.

The surface reconstruction results between the triple-dexel model and single-dexel model are also compared in this dissertation. Figure 2.6 illustrates the surface improvement from the triple-dexel data over the single-dexel data. Figures 2.6(a) and (c) show the results of surface reconstruction from single-dexel data, and Fig. 2.6(b) and (d) show the corresponding results of surface reconstruction from triple-dexel data. These figures clearly show that the generated surface from the triple-dexel data is more accurate

than the reconstructed surface from the single-dexel data when using the same ray resolution. In addition, to benchmark the performance of the developed method, numerical experiments are conducted to compare using triple-dexel data vs. voxel data in terms of the surface reconstruction time and the associated surface error. The test result shows that, under the same resolution, the surface reconstructed from the triple-dexel data has a smaller surface error in comparison with the surface reconstructed from the voxel data. This is because the triple-dexel based method utilizes actual positions of the intersection points between rays and the object's boundary surface as the vertices of the reconstructed surface model, while the voxel based method approximates the positions of these vertices by voxel interpolation.

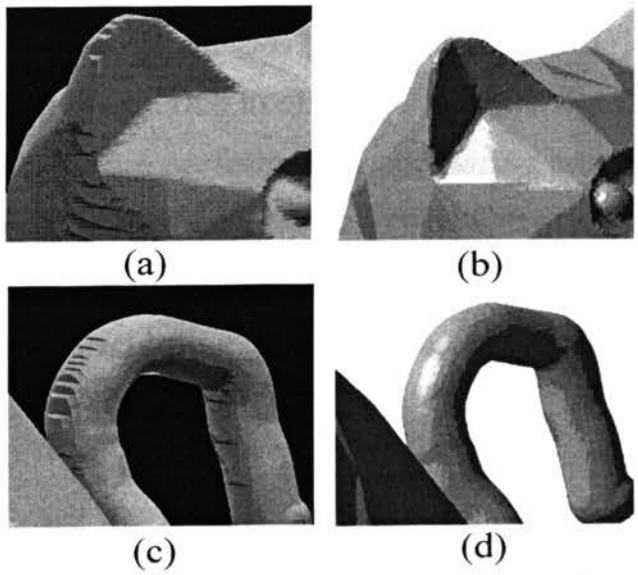


Figure 2.6. Comparisons of Reconstructed Surfaces. (a) and (c) are from Single-Dexel Data, (b) and (d) are from Triple-Dexel Data

The computation complexity of the contour generation, correspondence and combination process using triple-dexel data is $O(T)$ or $O(M^2)$, where M is the number of divisions along each axis. Because the complexity of the volume-based tiling algorithm is

also $O(M^2)$[Svitak and Skala, 2004], the developed surface reconstruction method is more efficient than the voxel-based surface reconstruction method, whose computational complexity is $O(M^3)$. Thus, the triple-dexel model is more efficient than the voxel model.

2.1.8. System Integration. The developed surface reconstruction process based on the triple-dexel model is incorporated into a virtual sculpting system [Peng and Leu, 2003, Leu et al., 2005, Peng et al., 2006]. The virtual sculpting system is developed on a Microsoft Windows XP workstation. The software is written in C++, and the graphics-rendering component is built on OpenGL and GLUT. The haptics interface is implemented using the PHANToM^{TM} device and the GHOST (General Haptics Open Software Toolkit) SDK software available from SensAble Technologies. This virtual sculpting system enables the user to create and modify 3D freeform objects through interactive sculpting operations and gives the user real-time force feedback during the sculpting process. The tool swept volume between two consecutive sampling times is obtained by the Sweep Differential Equation method [Blackmore Leu, 1992] and represented by boundary triangular meshes [Peng and Leu, 2003]. The workpiece and the tool swept volumes are scan-converted to obtain their triple-dexel data. Boolean operations on the triple dexels are performed by comparing and merging the dexel data in each of x, y or z directions. The surface reconstruction software is executed during the sculpting process to convert the triple-dexel model to a triangular mesh model. Figure 2.7 shows the setup of the virtual sculpting system and a cat model created using the system and viewed from two different directions.

2.2. STUDY OF DISTANCE FIELD BASED FREEFORM MODELING

The objective of this study is to develop more intuitive modeling operations for the virtual sculpting system such as the shape deformation and smoothing. The following tasks have been accomplished in this dissertation work.

2.2.1. Generation of Distance Field Model from Triple-Dexel Model. A four-step process is developed for generating the distance field. First the voxels that have non-null intersections with the solid's boundary surface are identified as the Boundary Voxels (BVs). The grid point on any edge of a BV is a Boundary Grid Point (BGP) and a grid

point is an Adjacent Grid Point (AGP) if it is adjacent to any BGP. Next, the sign of the distance value of each BGP and AGP is determined. Third, the surface within each BV is approximated using triangular facets. Finally, the distance value of each BGP and AGP is calculated. A 2D illustration is given in Fig. 2.8, where the gray-colored pixels surrounding the iso-surface are the boundary pixels (i.e., 2D BVs). Each squared point is a BGP and each triangular point is an AGP.

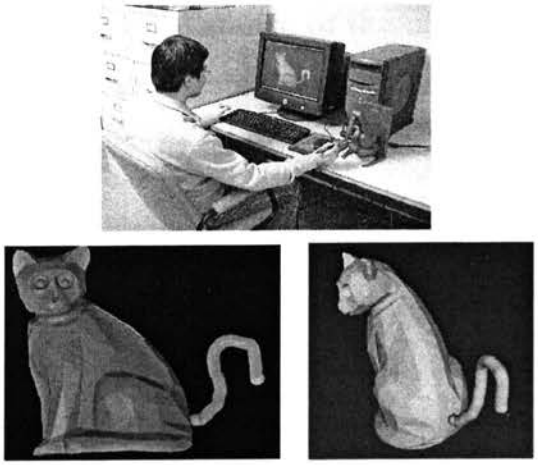


Figure 2.7. A Cat Model Generated Using the Virtual Sculpting System

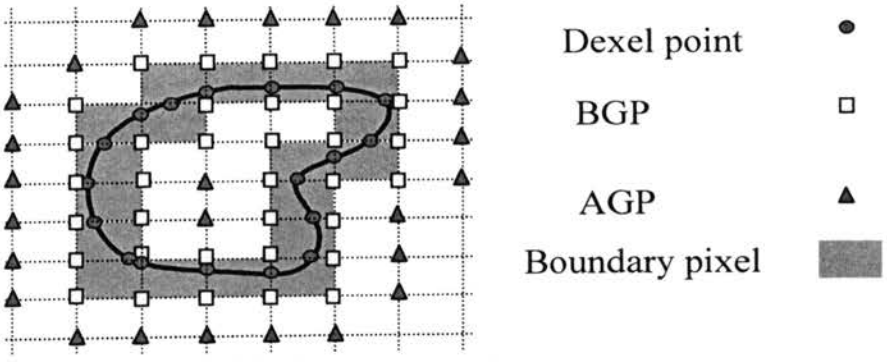


Figure 2.8. Boundary Pixels, BGP, AGP and Dexel Points

If a grid point is between two adjacent dexels along a ray, the distance value of this point is positive. Otherwise, the sign of the distance value is negative. The main idea of this step is to use the Hermite data (i.e., exact intersection points and normals) on the edges of a BV to calculate an additional point inside the BV by minimizing a quadratic function. By connecting this point with other additional points in adjacent BVs, triangular meshes can be generated with a simple patching algorithm to approximate the boundary surface. The Euclidean distance of a BGP of a BV is the shortest distance from the BGP to the local triangles formed by the additional point of this BV. The distances between this BGP and every such triangle are calculated, and the smallest value is the Euclidean distance. As illustrated in Fig. 2.9, the distance of the center grid point is d_2 because $d_2 < d_1$. Based on the same principle, to calculate the distance values of AGPs, such as point p_3 in Fig. 2.9, only triangles formed by the additional points in the adjacent BVs are considered for the distance test.

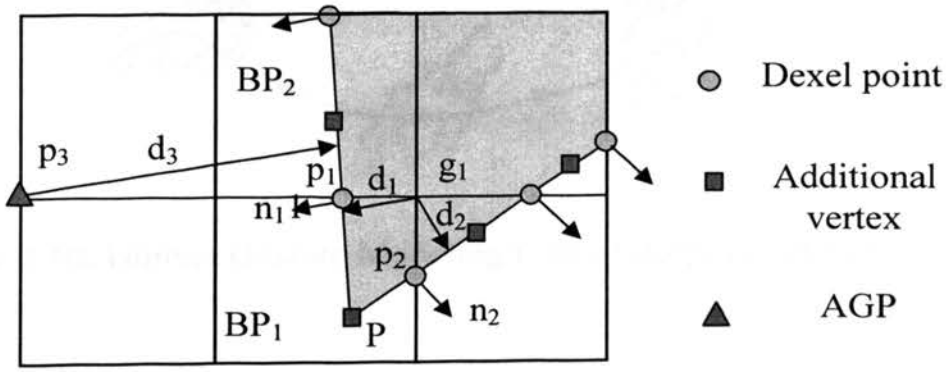


Figure 2.9. Distance Calculation for the Grid Points

2.2.2. Hand Gesture Modeling. A gesture is a form of non-verbal communication made with a part of the body such as the hand. The input of our freeform deformation framework is a series of gestures (i.e., orientations and positions of user's hand), G_i (i=0,…,n), captured from the mouse or 3D input devices such as the 6DOF tracking device. To associate user's gesture inputs with shape changes, the human gesture

has been modeled by formalizing a spatial transformation matrix. Then, freeform deformative operations are defined based on the human gesture model. Finally, a mapping method is developed to build connections between the defined operations and the boundary velocity of the surface which enables the level-set method to propagate the shape as desired. The gesture G_i at time t_i is defined by a local coordinate system with origin O_i and three orthogonal directions u_i, v_i, w_i as seen in Fig. 2.10, where $u_i \times v_i = 0$, $v_i \times w_i = 0$ and $w_i \times u_i = 0$. To produce a smooth space warp from input gestures, a B-Spline interpolation has been utilized to calculate the position and orientation of the gesture in between such as G_j in between G_i and G_{i+1} in Fig. 2.10. The gesture at G_j is constructed by the linear combination of translations and rotations around the interpolated origin O_j.

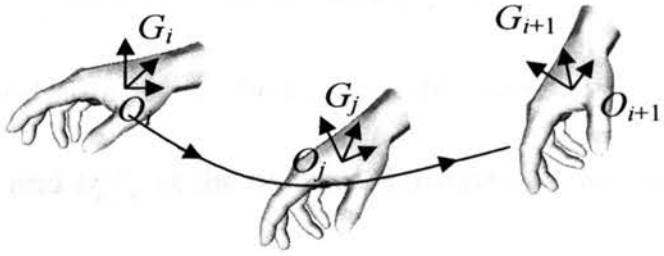


Figure 2.10. Human Gesture Modeling Using Interpolation Method

2.2.3. Shape Modeling Using Level-Set Method. Level-set models are deformable implicit surfaces where the deformation of the surface in its normal direction is controlled by a speed function in the level-set partial differential equation [Sethian, 1999]:

$$\frac{\partial F}{\partial t} = -\nabla F \cdot \bar{v} \tag{3}$$

where F(**x**,t) is the Euclidean distance function, **x** is the grid coordinates in Euclidean space R^3, $\bar{v}$ is the speed function of boundary points, ∇ is the gradient and

$$\nabla = i \cdot \frac{\partial}{\partial x} + j \cdot \frac{\partial}{\partial y} + k \cdot \frac{\partial}{\partial z} \tag{4}$$

where i, j and k are the unit vectors in R^3.

An up-wind scheme [Sethian, 1999] can be applied to resolve the level-set equation. The first-order space approximation of Equation (3) is:

$$F_{i,j,k}^{n+1} = F_{i,j,k}^{n} - \Delta t[\max(v_{i,j,k}^{n},0)\nabla^{+} + \min(v_{i,j,k}^{n},0)\nabla^{-}] \tag{5}$$

where $v_{i,j,k}$ is the speed at a point indexed by i, j and k and

$$\nabla^{+} = \begin{bmatrix} \max(D_{i,j,k}^{-x},0)^2 + \min(D_{i,j,k}^{+x},0)^2 + \\ \max(D_{i,j,k}^{-y},0)^2 + \min(D_{i,j,k}^{+y},0)^2 + \\ \max(D_{i,j,k}^{-z},0)^2 + \min(D_{i,j,k}^{+z},0)^2 \end{bmatrix}^{1/2} \tag{6}$$

$$\nabla^{-} = \begin{bmatrix} \max(D_{i,j,k}^{+x},0)^2 + \min(D_{i,j,k}^{-x},0)^2 + \\ \max(D_{i,j,k}^{+y},0)^2 + \min(D_{i,j,k}^{-y},0)^2 + \\ \max(D_{i,j,k}^{+z},0)^2 + \min(D_{i,j,k}^{-z},0)^2 \end{bmatrix}^{1/2} \tag{7}$$

where $D_{i,j,k}^{+x}$ is a shorthand notation of the forward difference operator $\frac{F_{i,j,k}(x+h,t) - F_{i,j,k}(x,t)}{h}$ and $D_{i,j,k}^{-x}$ is the backward difference operator $\frac{F_{i,j,k}(x,t) - F_{i,j,k}(x-h,t)}{h}$. The implementation of the level-set method can be speed up using a narrow-band scheme [Sethian, 1999]. The idea of this method is to update only a narrow-band of grid points which are close to the iso-surface rather than grid points in the entire space. The advantage of this approach is that the number of points being computed is so small that it is feasible to use a linked-list structure to keep track of them for real-time applications. By updating the distance values of the boundary grid points according to Eq. (5), the change of the iso-surface can be tracked.

2.2.4. Deformation Operation. Deformation operation imposes movements of surface vertices inside the influence region of a brush. At each time, these surface vertices p are adjusted by multiplying the weight $w(p)$ with the transformation matrix $t(p)$, and adding the result to the current value as:

$$p' = p + w(p) \cdot t(p) \tag{8}$$

To produce smooth transformation, the weight function can be defined as

$$w(x) = 1 - d^2(p)(3 - 2d(p)) \tag{9}$$

where $d(p)$ is the distance value of the point p to the center of the user's hand. To prorogate the shape using level-set method, the velocities of boundary grid points are defined according to the user's gesture inputs as follows: suppose a grid point p is on the surface, its transformed point under user gesture inputs is p'. Then the velocities of the grid points swept by pp' can be defined as $v=cpp'$ in the direction of vector pp' where c is a constant.

2.2.5. Smoothing Operation. If the speed (v) of a boundary point in Equation (3) is proportional to the user's hand motion and the mean curvature of the local boundary, then Equation (3) can be written as

$$\frac{\partial F(\mathrm{x},t)}{\partial t} - b(\mathrm{x},t)H(\mathrm{x},t) \| \nabla F(\mathrm{x},t) \| = 0 \tag{10}$$

where $b(\mathbf{x},t)$ is a transformation matrix defined by the user's gesture inputs and $H(\mathrm{x},t)$ is the mean curvature of point x. The mean curvature at a point $p \in S$ is the average of the principal curvatures (κ_1 and κ_2)

$$H = (\kappa_1 + \kappa_2)/2 \tag{11}$$

For a 3D surface defined as a function of three coordinates, e.g., $F(x,y,z)$, the mean curvature of a grid point is

$$H = \frac{(F_{yy} + F_{zz})F_x^2 + (F_{xx} + F_{zz})F_y^2 + (F_{xx} + F_{yy})F_z^2 - 2(F_xF_yF_{xy} + F_xF_zF_{xz} + F_yF_zF_{yz})}{2(F_x^2 + F_y^2 + F_z^2)^{3/2}} \tag{12}$$

where the differential terms are approximated using first-order, central finite difference, i.e,

$$F_x = \frac{F_{i+1,j,k} - F_{i-1,j,k}}{2\Delta x} \tag{13}$$

$$F_{xx} = \frac{F_{i+1,j,k} - 2F_{i,j,k} + F_{i-1,j,k}}{\Delta x^2} \tag{14}$$

$$F_{xy} = \frac{F_{i+1,j+1,k} - F_{i+1,j-1,k}}{4\Delta x \Delta y} + \frac{F_{i-1,j-1,k} - F_{i-1,j+1,k}}{4\Delta x \Delta y} \tag{15}$$

According to Eq. (10), the part of the boundary with larger curvature moves faster than the part of the boundary with smaller curvature in the surface normal direction. This movement results a smoothing operation.

2.2.6. Performance Evaluation. To evaluate the performance of the level-set method, a shrink operation is performed on a 2D circle shape. The number of grid points, the calculation time of distance values, and the time of updating the lists are given in Table 2.1. It can be seen form the table that a 10Hz refresh rate can be maintained by updating around 34,700 grid points for each iteration.

Table 2.1. Test Results of the Level-Set Method

No. of grid points	Time of calculating the distance values (s)	Time of updating the lists (s)	Total time (s)
202,592	0.4637	0.1631	0.6268
156,702	0.3675	0.0973	0.4648
149,942	0.3680	0.0902	0.4582
101,788	0.1754	0.0897	0.2651
28,260	0.0746	0.0108	0.0854
23,217	0.0638	0.0108	0.0746

3. MAJOR RESEARCH CONTRIBUTIONS

3.1. SURFACE RECONSTRUCTION FROM DEXEL MODELS

A novel method to convert dexel data into a series of planar contours on parallel slices has been developed. Comparing with other existing methods such as voxel based methods [Benouamer and Michelucci, 1997] and Delaunay based methods [Edelsbrunner and Mucke, 1994; Bernardini et al., 1999; Amenta et al., 2001; Dey et al., 2001], this method is faster and more efficient in terms of computational cost and memory usage. In addition, to our best knowledge, there has been no previous work on generating contours on three sets of orthogonal slices from triple-dexel data for the purpose of reconstructing a surface model. Thus, the developed surface reconstruction method is the first to reconstruct a triangular surface from triple-dexel data by using three orthogonal sets of contours. The main contributions of this research include: (i) creation of a methodology of surface reconstruction from triple-dexel data, (ii) development of a contour generation algorithm to create planar contours from dexel data, (iii) development of a contour combination algorithm to improve the accuracy of contours in representing the 3D model's cross sections, (iv) incorporation of a volume-based surface tiling algorithm in the surface reconstruction process, (v) complexity and accuracy analysis of the developed method, and (vi) benchmark with the voxel-based surface reconstruction method to demonstrate the efficiency of the developed method. The developed surface reconstruction method provides a good solution to the view-dependent problem inherent in dexel model. The method has been applied to different real-time applications such as virtual sculpting [Zhang et al., 2007; Zhang and Leu, 2008b] and NC machining simulation [Zhang and Leu, 2008a]. A formal analysis has been performed on the computational complexities of the develop algorithms in order to evaluate their performance [Zhang et al., 2005]. Details descriptions of the developed methods for surface reconstruction from dexel data are presented in the *first two papers* included in this dissertation.

3.2. DISTANCE FIELD GENERATION FROM TRIPLE-DEXEL MODEL

A brute force method is generally used to compute the distances from a grid point in the Euclidean space to every boundary triangle of M and select the shortest one. To reduce the computation, the shortest distance can be calculated only to a limited number of primitives according to spatial coherences. There has been very little research on the calculation of the distance field directly from triple-dexel data for the generation of a triangular mesh in virtual sculpting. Sealy and Novins [1999] approximated the Euclidean distance of a grid point as the shortest distance among its three axial distances. But this approximation is not accurate especially where sharp features are present. In this dissertation, the distance field data is firstly generated from triple-dexel data by approximating the iso-surface inside the boundary voxels and calculating the Euclidean distance values for a narrow-band of grid points. This method is capable of generating a more accurate distance field since distance value is calculated as the shortest distance from a grid point to the boundary surface inside of each cell [Zhang and Leu, 2008c]. Details of the distance field generation from triple-dexel data are presented in the *third paper* included in this dissertation.

3.3. LEVEL-SET METHOD BASED FREEFORM OPERATIONS

The level-set method [Sethian, 1999] provides mathematical and numerical mechanisms for computing surface deformations as time-varying iso-values of a function by solving a partial differential equation on the 3D grid. A set of numerical techniques is provided by the level-set formulation that describes how to manipulate the distance values of each grid in a volume, so that the iso-surfaces of the function move in a prescribed manner. Previous studies in the field of level-set method based freeform geometric modeling focused on developing various surface editing operators such as blending, smoothing, sharpening, opening/closings, and embossing [Museth et al., 2002, 2005; Bærentzen and Christensen, 2002; Lawrence and Funkhouser, 2004]. None of the previous work modeled human gestures and developed gesture based freeform modeling operations based on the level-set method. In this study, the modeling of human hand

gestures has been developed and utilized to define various freeform modeling operations such as sculpting, imprint, deformation and smoothing.

Using gesture information for the freeform modeling provides unique tools for freeform modeling since it is more natural to the user's design intent. In addition, level-set models offer several advantages in geometric modeling than the traditional mesh-based modeling framework where the shape is represented by triangular meshes; they include: 1) by construction, self-intersection cannot occur when using the level-set method. This guarantees the generation of physically-realizable, simple, closed surfaces. 2) Level-set model can easily change topological genus, and 3) the generated models are free of edge connectivity and mesh quality problems which are associated with mesh models.

In this study, the gesture of the user is modelled by the B-Spline interpolation and the linear combination of user's hand inputs. Deformation, imprint, and smoothing operations have been developed. After mapping the velocities of boundary grid points for each operation, the solution of the level-set method drives the propagation of the shape towards the desired shape. Comparing with the traditional mesh based method, the triangular meshes generated using the level-set methods developed this paper are free of the self-intersection problem [Zhang and Leu, 2007, 2008d]. Details of the development of generic freeform modeling operations based on the level-set framework are presented in the *fourth paper* included in this dissertation.

BIBLIOGRAPHY

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T., "Point Set Surfaces," IEEE Visualization 2001, San Diego, California, October 21-26, pp. 21–28.

Amenta, N., Choi, S., and Kolluri, R., 2001, "The Power Crust," in 6th ACM Symposium on Solid Modeling and Applications, Ann Arbor, MI, June 4-8, pp. 249 – 266.

Azernikov, S., Miropolsky, A., and Fischer, A., 2003, "Surface Reconstruction of Freeform Objects Based on Multi-resolution Volumetric Method," Proc. of ACM Symposium on Solid and Physical Modeling, Seattle, Washington, June 16 - 20 pp. 115-126.

Bajaj, C., Lin, K. and Coyle, E., 1996, "Arbitrary Topology Shape Reconstruction from Planar Cross-sections," Graphic Models and Image Processing, 58, pp.524-543.

Bærentzen, J. A., and Christensen, N. J., 2002, "Volume Sculpting Using The Level-Set Method," Proc. of Shape Modeling International'02, pp. 175-182.

Beier, T., 1993, "Practical Uses for Implicit Surfaces in Animations," Modeling, Visualizing and Animation Implicit Surfaces (SIGGRAPH'93 Course Notes), pp. 20.1-20.10.

Benouamer, M. O., and Michelucci, D., 1997, "Bridging the Gap between CSG and Brep via a Triple Ray Representation," Solid Modeling'97 Atlanta, GA, pp. 68-79.

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G., 1999, "The Ball-Pivoting Algorithm for Surface Reconstruction," IEEE Trans. Vis. Comput. Graph., 5(4), pp. 349–359.

Blackmore, D., and Leu, M.C., 1992, "Analysis of Swept Volume via Lie Groups and Differential Equations," International Journal of Robotics Research, pp. 516-537.

Blinn, J. F., 1982, "A Generalization of Algebraic Surface Drawing," ACM Trans. on Graphics, 1(3), pp. 235-256.

Bloomenthal J., and Wyvill, B., 1990, "Interactive Techniques for Implicit Modeling," Computer Graphics (1990 Symp. On Interactive 3D Graphics), 24(2), pp. 109–116.

Bloomenthal, J., editor, 1997, Introduction to Implicit Surfaces, Morgan Kaufmann Publishers, Inc., San Francisco, California.

Boissonnat, J.D., 1988, "Surface Reconstruction from Planar Cross-sections," Computer Vision Graphics and Image Processing, 44, pp.1-29.

Butterworth, J., Davidson, A., Hench, S., and Olano, M., 1992, "3DM: A Three Dimensional Modeler Using a Head-Mounted Display," Proceedings of Symposium on Interactive 3D Graphics, pp. 135-138.

Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R., 2001, "Reconstruction and Representation of 3D Objects with Radial Basis Functions," Proc. of the 28th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH, Los Angeles, California, August 12-17, pp. 67-76.

Cheng, S.W. and Dey, T.K., 1988, "Improved Constructions of Delaunay Based Contour Surfaces," Proceedings of the fifth ACM Symposium on Solid modeling and Applications, pp.322-323.

Chen, M., Kaufman, A., and Yagel, R., Eds. 2001, Volume Graphics, Springer-Verlag New York, Inc.

Christiansen, H. N., and Sederberg, T. W., 1978, "Conversion of Complex Contour Line Definitions into Polygonal Element Mosaics," Proc. of the 5th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, pp. 187-192.

Cignoni, P., Montani, C., and Scopigno, R., 1998, "A Comparison of Mesh Simplification Algorithms," Computer Graphics, 22(1), pp. 37–54.

Coquillart, S., 1990, "Extended Freeform Deformation: A Sculpturing Tool for 3D Geometric Modeling," Computer Graphics, 24(4), pp. 187-196.

Cornea N.D., Silver D., and Min P., 2005, "Curve-Skeleton Applications," Proceedings IEEE Visualization, pp. 95-102.

Crespin, B., Blanc, C., and Schlick, 1996, "Swept Implicit Surface," Computer Graphics Forum, 15(3), pp. 165-174.

Crossno, P., and Angel, E., 1997, "Iso-surface Extraction using Particle Systems," Proceedings of the 8th Conference on Visualization, pp. 495–505.

Dani, T.H. and Gadh, R., 1997, "Creation of Concept Shape Designs via a Virtual Reality Interface," Computer-Aided Design, 29(8), pp.555-563.

Deering, M.F., 1996, "The Holosketch VR Sketching System," Communications of the ACM, 39(5), pp. 54-61.

Dey, T., Giesen, J., and Hudson, J., 2001, "A Delaunay Based Shape Reconstruction From Large Data," Proc. of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics, San Diego, California, Oct 22-23, pp. 19-27.

Doi, A., and Koide, A., 1991, "An Efficient Method of Triangulating Equivalued Surfaces by using Tetrahedral Cells," IEICE Transactions Communication, Elec. Info. Syst, 74(1), pp. 214-224.

Edelsbrunner, H., and Mucke, E. P., 1994, "Three-dimensional Alpha Shapes," ACM Transactions on Graphics, 13(1), pp. 43–72.

Felkel, P. and Obdržálek, S., 1999, "Improvement of Oliva's Algorithm for Surface Reconstruction from Contours," Proceedings of the 15th Spring Conference on Computer Graphics, Budmerice, Slovakia, pp. 254-263.

Frisken, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R., 2000, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics," Proceeding of SIGGRAPH 2000, pp. 249-254.

Fuchs, H., Kedem, Z.M. and Uselton, S.P., 1977, "Optimal Surface Reconstruction from Planar Contours," Communications of the ACM, 20(10), pp.693-702.

Gagvani, N., and Silver, D., 2001, "Animating Volumetric Models," Academic Press Professional, 63(6), pp. 443-458.

Galin, E., Leclercq A., and Akkouche, S., 1999, "Blob-Tree Metamorphosis," Implicit Surfaces'99 Conference, Bordeaux, France.

Hilton, A., Stoddart, A.J., Illingworth, J., and Windeatt, T., 1996, "Marching Triangles: Range Image Fusion for Complex Object Modeling," In International Conf. on Image Processing, pp. 381-384.

Hsu, W. M., Hughes, J. F. and Kaufman, H., 1992, "Direct Manipulation of Freeform Deformation," Computer Graphics, 26(2), pp. 177-184.

Huang, Y., and Oliver, J. H., 1995, "Integrated Simulation, Error Assessment and Tool Path Correction for Five-axis NC Milling," Journal of Manufacturing Systems, 14(5), pp. 331-344.

Igarashi, T., Matsuoka, and S., Tanaka, H., 1999, "Teddy: A Sketching Interface for 3D Freeform Design," ACM SIGGRAPH 1999, pp. 409-417.

Keppel, E., 1975, "Approximating Complex Surfaces by Triangulation of Contour Lines," IBM Journal of Research and Development, 19(1), pp.2-11.

Kirkpatrick, D.G., and Radke J.D., 1985, "A Framework for Computational Morphology," in Computational Geometry, Toussaint, G. T., eds., North-Holland, pp. 217-248.

Kobbelt, L. P., Botsch, M., Schwanecke, U., and Seidel, H. 2001, "Feature Sensitive Surface Extraction from Volume Data," Proc. of the 28th Annual Conference on Computer Graphics and interactive Techniques, SIGGRAPH '01, ACM, New York, NY, pp. 57-66.

Konig, A. H., and Groller, E., 1998, "Real-Time Simulation and Visualization of NC Milling Processes for Inhomogeneous Materials on Low-End Graphics Hardware," Proc. of the Computer Graphics International, Hannover, Germany, June 22–26, pp. 338.

Lamousin, H. J., and Waggenspack, W. N., 1994, "NURBS-Based Freeform Deformations," IEEE Computer Graphics & Applications, pp 59-65.

Lawrence, J. and Funkhouser, T., 2004, "A Painting Interface for Interactive Surface Deformations," Graph. Models 66(6), pp. 418-438.

Leu, M. C., Maiteh, B. Y., Blackmore, D., and Fu, L., 2001, "Creation of Freeform Solid Models in Virtual Reality," Annals of the CIRP, 50(1), pp. 73-76.

Liang, J., and Green, M., 1994, "JDCAD: A Highly Interactive 3D Modeling System," Computers and Graphics, 18(4), pp. 499-506.

Lieutier, A., 2004, "Any Open Bounded Subset of Rn Has the Same Homotopy Type than Its Medial Axis," Computer-Aided Design, 36, pp. 1029-1046.

Lorensen, W.E., and Cline, H.E., 1987, "Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm," Computer Graphics, 21(4), pp. 163-169.

Jones, M. W., Baerentzen, J. A., and Sramek, M., 2006, "3D Distance Fields: A Survey of Techniques and Applications," IEEE Transactions on Visualization and Computer Graphics, 12(4), pp. 581-599.

Klein, R., Schilling, A. and Straßer, W., 1999, "Reconstruction and Simplification of Surfaces from Contours," In Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications, pp.198-207.

Maiteh, B. Y., Blackmore, D., Abdel-Malek, L., and Leu, M. C., 2000, "Swept-Volume Computation for Machining Simulation and Virtual Reality Application," Journal of Materials Processing and Manufacturing Science, 7, pp. 380-390.

Maragos, P., 1996, "Differential Morphology and Image Processing," IEEE Trans. on Image Processing, 5(6), pp. 922–937.

Maya, Alias, 2006, http://www.alias.com.

Meyers, D., Skinner, S. and Sloan, K., 1992, "Surfaces from Contours," ACM Transactions on Graphics, 11(3), pp.228-258.

Muller, H., Surmann, T., Stautner, M., Albersmann, F., and Weinert K., 2003, "Online Sculpting and Visualization of Multi-Dexel Volumes," SM'03, Seattle, Washington, June 16-20, pp. 258-261.

Museth, K., Breen, D. E., Whitaker, R. T., and Barr, A. H., 2002, "Level-Set Surface Editing Operators", SIGRAPH'2002, pp. 330-338.

Museth, K., Breen, D. E., Whitaker, R. T., Mauch, S., and Johnson, H., 2005, "Algorithms for Interactive Editing of Level-Set Models," Computer Graphics Forum, 24(4), pp. 821-841.

Nathan, J., 1999, SONY: The Private Life, Houghton Mifflin Company, New York, NY.

Nielsen, M. B., Museth, K., 2006, "Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level-Sets," Journal of Scientific Computing, 26(3), pp. 1573-7691.

Nishimura, H., Hirai, A., Kawai, T., Kawata, T., Shirakawa, I., and Omura, K., 1985, "Object Modeling by Distribution Function and a Method of Image Generation," Journal of papers given at the Electronics Communication Conference '85.

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H. 2003, "Multi-Level Partition of Unity Implicits," ACM Trans. Graph., 22(3), pp. 463-470.

Ohtake, Y., Belyaev, A., and Seidel, H. 2006, "Sparse Surface Reconstruction with Adaptive Partition of Unity and Radial Basis Functions," Graph. Models, 68(1), pp. 15-24.

Oliva, J.M., Perrin, M. and Coquillart, S., 1996, "3D Reconstruction of Complex Polyhedral Shapes from Contours Using a Simplified Generalized Voronoi Diagram," Computer Graphics Forum, 15(3), pp.397-408.

Parent, R., 1977, "A System for Sculpting 3-D Data," Computer Graphics, 11(2), pp. 138-147.

Peng X. and Leu, M.C., 2005, "Engineering Applications of Virtual Reality," Mechanical Engineer's Handbook, 3rd edition, John Wiley and Sons.

Peng, X., Zhang, W., Asam, S., and Leu, M.C., 2004, "Surface Reconstruction from Dexel Data for Virtual Sculpting," Proceedings of ASME International Mechanical Engineering Conference, Anaheim, CA, November 14-19.

Peng, X., Zhang, W., and Leu, M. C., 2006, "Freeform Modeling Using Sweep Differential Equation with Haptic Interface," Journal of Virtual and Physical Prototyping.

Ren, Y., Lai-Yuen, K. S., and Lee, Y. S., 2006, "Virtual Prototyping and Manufacturing Planning by Using Tri-dexel Models and Haptic Force Feedback," Virtual and Physical Prototyping, 1(1), pp. 3-18.

Rosch, A., Ruhl, M., and Saupe, D., 1996, "Interactive Visualization of Implicit Surfaces with Singularities," Computer Graphics Forum, 16(5), pp. 295–306.

Sachs, E., Roberts, A. and Stoops, D., 1991, "3-Draw: A Tool for Designing 3D Shapes," In Proceedings of IEEE Computer Graphics and Applications, pp. 18-24.

Sapiro, G., Kimmel, R., Shaked, D., Kimia, B., and Bruckstein, A., 1993, "Implementing Continuous-scale Morphology via Curve Evolution," Pattern Recognition, 9, pp. 1363–1372.

Sealy, G. and Novins, K., 1999, Effective Volume Sampling of Solid Models using Distance Measures, Proc. of the international Conference on Computer Graphics.

Sederberg, T. W., Parry, S. R., 1986, "Freeform Deformation of Solid Geometric Models," Computer Graphics, 20(4), pp. 151-160.

SensAble Technologies, 2006, Freeform® Concept, www.sensable.com/products/3ddesign.

Sethian, J. A., 1999, Level-Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science (2nd Ed.), Cambridge University Press, UK.

Sloan, K.R. and Painter, J., 1987, "From Contours to Surfaces: Test bed and Initial Results," In Proceedings of CHI+GI '87, pp.115-120.

Svensson, S., Nystrom. I., and Sanniti di Baja G., 2002, "Curve-skeletonization of Surface-Like Objects in 3D Images Guided by Voxel Classification," Pattern Recognition Letters, 23(12), pp. 1419-1426.

Svitak, R., and Skala, V., 2004, "A Robust Technique for Surface Reconstruction from Orthogonal Slices," Machine Graphics & Vision, 12(3), pp. 221-233.

Treece, G.M., Prager, R.W., Gee, A.H., and Berman, L., 2000, "Surface Interpolation from Sparse Cross-Sections Using Region Correspondence," IEEE Transactions on Medical Imaging, 19(11), pp.1106-1114.

Turk, G., and Brien, J. F., 2002, "Modeling with implicit surfaces that interpolate," ACM Transactions on Graphics, 21(4), pp. 855-873.

Wang, S., and Kaufman, A., 1995, "Volume Sculpting," Symposium on Interactive 3D Graphics, ACM Press, pp.151-156.

Wang, Y. F., and Aggarwal, J. K., 1986, "Surface Reconstruction and Representation of 3D Scenes," Pattern Recognition, 19(3), pp. 197-207.

Ware, C., and Jessome, D.R., 1988, "Using the Bat: a Six Dimensional Mouse for Object Placement," Proceedings of Graphics Interfaces, pp. 119-124.

Witkin A. P., and Heckbert, P. S., 1994, "Using Particles to Sample and Control Implicit Surfaces," International Conference on Computer Graphics and Interactive Techniques, pp. 269-277.

Wyvill, B., Galin, E., and Guy, A., 1999, "Extending the CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System," Computer Graphics Forum, 18(2), pp. 149–158.

Wyvill, G., McPheeters, C., and Wyvill, B., 1986, "Data Structure for Soft Objects," The Visual Computer, 2(4), pp. 227–234.

Van Hook, T., 1986, "Real Time Shaded NC Milling Display," Computer Graphics, 20(4), pp. 15-20.

Veltkamp, R.C., 1992, "The γ-neighborhood Graph," Computational Geometry: Theory and Applications, 1(4), pp. 227-246.

Zhang, W., and Leu, M. C., 2007, Interactive Sketch-Based Digital Prototyping by Using the Level-Set Method, Proc. of ASME International Mechanical Engineering Congress and Exposition, November 11-15, Seattle, Washington.

Zhang, W., and Leu, M. C., 2008a, NC Machining Simulation Using Triple-Dexel Representation, 2008 International Symposium on Flexible Automation (ISFA), June 23–26, Atlanta, Georgia.

Zhang, W., and Leu, M. C., 2008b, Surface Reconstruction Using Dexel Data From Three Sets of Orthogonal Rays, ASME Journal of Computing and Information Science in Engineering, 8(3), September.

Zhang, W., and Leu, M.C., 2008c, Virtual Sculpting with Surface Smoothing Based on the Level Set Method, Annuals of the CIRP, 57(1), pp. 167-170.

Zhang, W., and Leu, M.C., 2008d, Gesture Based Freeform Geometric Modeling Using Level-Set Method, 2nd Annual ISC Research Symposium (ISCRS 2008), April 22, Rolla, Missouri.

Zhang, W., Peng, X., Leu, M. C., and Blackmore, D., 2005, Accuracy and Computational Complexity Analysis of Design Models Created by Virtual Sculpting, Proc. of ASME International Mechanical Engineering Congress and Exposition, November 5-11, Orlando, Florida.

Zhang, W., Peng, X., Leu, M. C., and Zhang, W., 2007, A Novel Contour Generation Algorithm for Surface Reconstruction from Dexel Data, ASME Journal of Computing and Information Science in Engineering, 7(3), pp. 203-210.

Zhu, W., and Lee, Y.S., 2005, "A Visibility Sphere Marching Algorithm of Constructing Polyhedral Models for Haptics Sculpting and Product Prototyping," International Journal of Robotics and Computer Integrated Manufacturing, 21(1), pp. 19-36.

Zhu, W., 2003, "Virtual Sculpting and Polyhedral Machining Planning System with Haptic Interface," Ph.D. Thesis, North Carolina State University, http://www.lib.ncsu.edu/theses/available/etd-08172003-194602/

Zorriassatine, F., Wykes, C., Parkin, R., and Gindy, N., 2003, "A Survey of Virtual Prototyping Techniques for Mechanical Product Development," Journal of Engineering Manufacture, 217(4), pp. 513-530.

PAPER

I: A NOVEL CONTOUR GENERATION ALGORITHM FOR SURFACE RECONSTRUCTION FROM DEXEL DATA

Weihan Zhang[1], Xiaobo Peng[2], Ming C. Leu[1], Wei Zhang[3]

[1]Department of Mechanical and Aerospace Engineering
University of Missouri-Rolla
Rolla, Missouri 65409, USA
Email: wzxq6@umr.edu

[2]Mechanical Engineering Department
Prairie View A&M University
Prairie View, TX 77446, USA
Email: xipeng@pvamu.edu

[3]Department of Industrial Engineering
Tsinghua University
Beijing, 100084, P. R. CHINA
Email: zhangwei@tsinghua.edu.cn

ABSTRACT

This paper presents a method of reconstructing a triangular surface patch from dexel data generated by ray casting, to represent solid models for applications such as virtual sculpting and NC machining simulation. A novel contour generation algorithm is developed to convert dexel data into a series of planar contours on parallel slices. The algorithm categorizes the dexels on two adjacent rays into different groups by using a "grouping" criterion. The dexel points in the same group are connected using a set of rules to form sub-boundaries. After checking the connections among all the dexel points on one slice, a connection table is created and used to obtain the points of connection in a counterclockwise sequence for every contour. Finally, the contours on all the parallel slices are tiled to obtain triangular facets of the boundary surface of the 3D object. Computational costs and memory requirements are analyzed, and the computational complexity analysis is verified by numerical experiments. Example applications are given to demonstrate the described method.

Keywords:
Dexel Representation, Contour Generation, Surface Reconstruction, Simulation

1. INTRODUCTION

The dexel representation of a solid consists of a set of line segments lying inside the solid. These segments are obtained by classifying a grid of parallel rays, a process often called ray casting or ray tracing [1, 2]. As illustrated in Fig. 1, for each ray the intersection points with the solid are stored in the following manner: two points defining a line segment that is fully inside the solid make up a dexel. Each dexel has two end points (known as dexel points) and referred to as the head and the tail (the order of which defines the direction). Dexels may also contain tags (i.e. attributes), which are symbolic data associated with each line segment representing material or other properties of the interior of a solid.

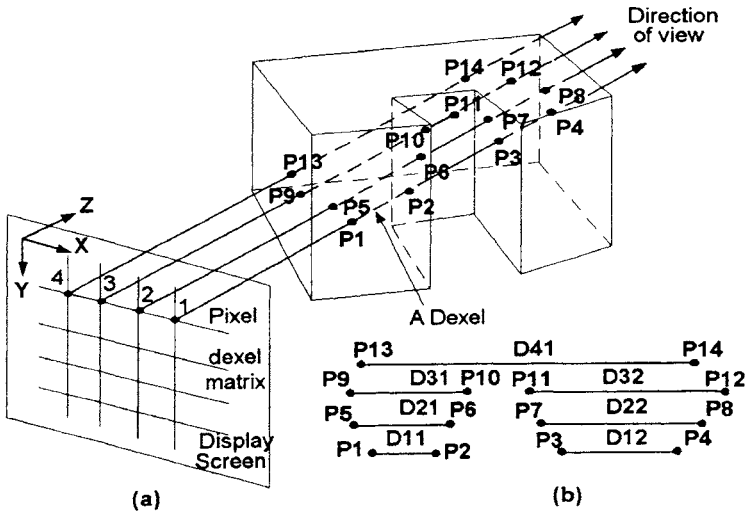


Figure 1. Illustration of the Ray Casting Process and the Dexel Representation

The dexel representation is an approximate representation method. In some applications such as NC machining simulation, more accurate representations such as the

constructive solid geometry (CSG) and the boundary representation (B-rep) are not suitable because Boolean operations involved in these representations are very time consuming and would require the use of paralleled algorithms and associated multi-processors hardware to speed up these processes for purpose of real-time implementation [3]. Approximate representation methods also include vector clipping [4], Z-map [5], G-buffer models [6], contour models [7] and voxel models [8]. A literature review about approximate representations for NC machining simulation was given by Erik et al. [9].

The dexel representation and its variations are among the most notable approximate representations used to support machining simulation because they allow fast Boolean operation, need little memory, have simple data structures, and have robust algorithms for development of real-time simulation applications. These advantages were evident when Van Hook [10] developed a real-time shaded display of a solid model being milled by a NC cutting tool. The dexel representation approach was also used by Konig and Groller [11] in their NC simulation work, which achieved real-time simulation and visualization for removal of inhomogeneous materials on low-end graphics hardware. Muller et al. [12] presented the idea of using multi-dexel volumes (with dexels generated by rays in multiple directions) to represent a solid in NC simulation. Ren et al. [13] developed a multi-dexel based machining planning system. Leu and his associates [14-16] developed a dexel-based system for design of parts with freeform geometry by virtual sculpting.

Challenging open problems still remain of the common dexel representation method due to the fixed direction in the ray casting process. Dexel data is view-dependent because it only records the geometric information of a 3D object from one viewing direction, as seen in Fig. 1. In the practice of dexel-based NC simulation, researchers were only able to produce a limited number of views from certain directions for the simulation, without the generation of a surface model that can be viewed from any directions. To solve the view-dependent problem, Huang and Oliver [17] briefly described a contour tracking technique but without detailed development of an algorithm. They visualized the boundary of the object by simply displaying sets of contours extracted from the dexel data. Konig and Groller [11] described an algorithm to create a surface representation from dexel data for 3-axis milling simulation. But the algorithm could fail easily in the virtual sculpting process where dexel data are modified in arbitrary

directions. Zhu and Lee [18] presented a visibility sphere marching algorithm for constructing polyhedral models from dexel data for their virtual sculpting research. When the algorithm was applied to complex 3D objects, there could be some cracks and holes in the generated mesh due to topology related issues [19]. The Marching Cube Algorithm [20] has been used to generate an approximate triangular surface from tri-dexel data [21] and from voxel data. But this algorithm requires huge memory storage and suffers from some ambiguity, and it can not be applied to dexel data generated in a single direction.

Another line of related research is the curve reconstruction study in computational geometry stated as follows: given a set of sample points from a curve, a reconstruction of the curve is intended, i.e., points are to be joined by edges in the order they appear on the curve. The dexel points can be seen as the points on the curves in relation to this study. The developed methods included the α-shape [22], β-skeleton [23], and γ-neighborhood graph [24]. But all of them require certain preconditions on the input points. The α-shape method works well for the points which are evenly distributed in the interior of an object. The β-skeleton method requires the sampling density of points varied with the local feature size on the curve. These curve reconstruction methods can not be directly applied to dexel data due to the nature of their input data.

2. CONTOUR GENERATION FROM DEXEL DATA

2.1. Algorithm Design Methodology

If the dexel data are sampled from a slice with one single closed contour, connecting the dexel points to form the contour is relative easy. Difficulties arise when there exist inner contours on slices taken from a 3D model with interior voids. Our approach to address the inner-contour difficulty is to design an algorithm that dictates how to connect dexel points on two adjacent rays for any considered planar slice by separating the dexels into groups. This requires the development of a grouping criterion, which categories the dexels on two adjacent rays into different groups. The main idea behind our design of the grouping criterion is the observation that two overlapping dexel spaces on two adjacent rays may form part of an inner contour.

As an illustration of this observation, one slice of the 3D model on XZ plane in Fig. 2(a) has dexel data shown in Fig. 2(b). The overlapping dexel spaces between points 6

and 7 and between points 12 and 13 form an inner contour because the top of these overlapped dexel spaces is covered by dexel B and the bottom is covered by dexel A.

According to this observation, if a set of overlapping dexel spaces is covered by both a dexel beneath the bottom and a dexel right above the top, these dexel spaces form an inner contour and are called *a closed set*. For example, the set of dexel spaces between points 6 and 7 and between points 12 and 13 in Fig. 2(b) is a closed set. The connections of a closed set of dexel spaces to form an inner contour are: filling the top and the bottom dexel spaces, and connecting the boundary dexel points on the same side of the dexel spaces accordingly (e.g. connecting point 6 and point 12, and connecting point 7 and point 13 in Fig. 2(b)).

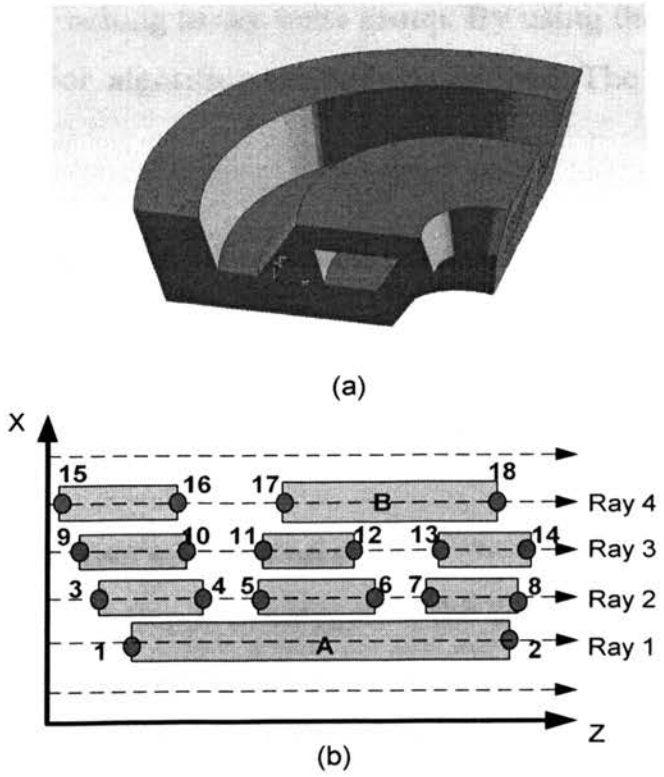


Figure 2. Example of the Contour Generation Algorithm. (a) 3D Model (b) One Slice of the 3D Model on XZ Plane

Meanwhile, if a set of overlapping dexel spaces is not covered by both a dexel beneath the bottom and a dexel above the top, it is an open set of dexel spaces. Their dexel points need to be connected differently to form part of an outer contour. For example, the dexel spaces between points 4 and 5, between points 10 and 11, and between points 16 and 17 in Fig. 2(b) are an open set. The connections of an open set of dexel spaces are: filling the top or the bottom dexel space, depending on which is covered by a dexel above or beneath, and connecting the boundary dexel points on the same side of the dexel spaces accordingly.

Based on the above discussion, the grouping criterion is defined as follows: two dexels on two adjacent rays of a planar slice belong to the same "group" if they overlap with each other. The grouping criterion represents a transitive relation ($\otimes$) over the dexel set X as follows: $\forall m,n,o \in X$, if $(m \otimes n) \wedge (n \otimes o)$, then $m \otimes o$; meaning that for three dexel spaces m, n and o, if m and n belong to the same group and n and o belong to the same group, then m and o belong to the same group. By using the grouping criterion, a four-step contour generation algorithm has been developed. The details are presented in the following.

2.2. Algorithm Details

For ease of discussion, the ray direction is assumed in the Z direction, which is also the dexel direction. The contour generation algorithm starts from the left-most dexel on the first ray (dexel A in Fig. 2(b)) intersected with the object. It continues to increase the ray number by one in X direction, and ends at the right-most dexel on the last ray (dexel B in Fig. 2(b)).

a) Step 1: Group dexels on two adjacent rays

The objective of the first step is to categorize the dexels on every two adjacent rays into groups according to the grouping criterion. Two sets of dexels on Rays i and Ray $i+1$ are taken as the input and separated into a number of dexel groups $Ng_{i,p}$, $i \in [1, RR]$, $p \in [1, N_i]$, where p is the group index, RR is the total number of rays intersecting with the object on the slice, and N_i is the total number of groups between Ray i and Ray $i+1$. For example, in Fig. 3, after the first step based on the defined grouping criterion, two groups are identified: the first group consists of $D_{i+1,1}$, $D_{i+1,2}$, $D_{i,1}$, $D_{i,2}$ and $D_{i,3}$, and the second group consists of $D_{i+1,3}$ and $D_{i,4}$. They are shown in different patterns and colors.

b) Step 2: Connect adjacent dexel points inside each group

The aim of step 2 is to generate connections between dexel points in the same group along every two adjacent rays. Suppose a group of dexels consists of R_i dexels ($D_{i,1}$, $D_{i,2}$, . . . $D_{i,Ri}$) on Ray i and R_{i+1} dexels ($D_{i+1,1}$, $D_{i+1,2}$, . . . $D_{i+1,Ri+1}$) on Ray i+1, where $R_i \geq 1$ and $R_{i+1} \geq 1$, as illustrated in Fig. 4. $D_{i,j}\rightarrow[h]$ and $D_{i,j}\rightarrow[t]$ are the head and the tail of dexel $D_{i,j}$, respectively. The first two dexel points $D_{i,1}\rightarrow[h]$ and $D_{i+1,1}\rightarrow[h]$ should be connected because they are two adjacent points on the same outer boundary. Likewise, the last two dexel points $D_{i,Ri}\rightarrow[t]$ and $D_{i+1,Ri+1}\rightarrow[t]$ are also connected. The points in between should be connected to the adjacent dexel points on the same ray. Thus, the rules of connections within a dexel group are:

①: Connect ($D_{i+1,1}\rightarrow[t]$, $D_{i+1,2}\rightarrow[h]$), . . . ($D_{i+1,\,Ri+1-1}\rightarrow[t]$, $D_{i+1,Ri+1}\rightarrow[h]$)

②: Connect ($D_{i,1}\rightarrow[t]$, $D_{i,2}\rightarrow[h]$), . . . ($D_{i,\,Ri-1}\rightarrow[t]$, $D_{i,Ri}\rightarrow[h]$)

③: Connect ($D_{i,1}\rightarrow[h]$, $D_{i+1,1}\rightarrow[h]$)

④: Connect ($D_{i,Ri}\rightarrow[t]$, $D_{i+1,Ri+1}\rightarrow[t]$)

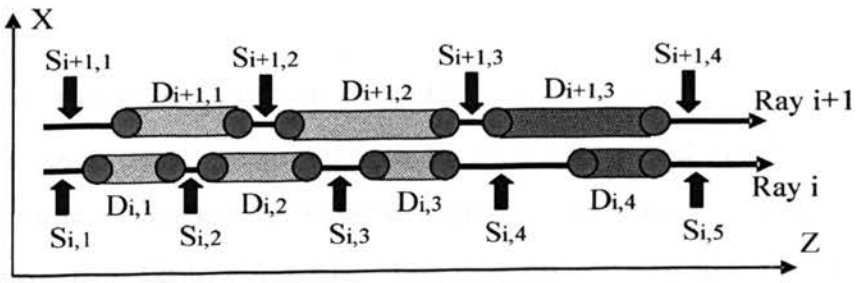


Figure 3. Grouping Process

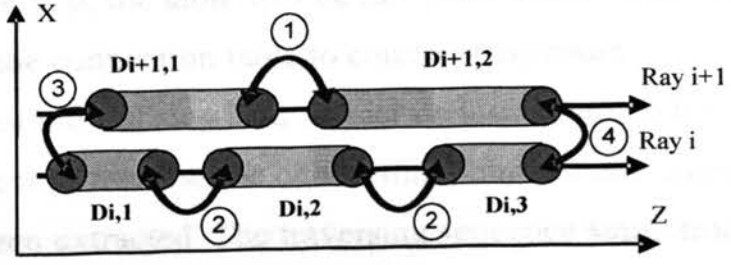


Figure 4. Contouring Algorithm

Special cases exist when one of the two adjacent rays does not intersect with the 3D object, as shown in Fig. 5. The rules of connections for these cases are:

⑤: When $R_i = 0$, connect ($D_{i+1,1}$→[h], $D_{i+1,1}$→[t])

⑥: When $R_{i+1} = 0$, connect ($D_{i,1}$→[h], $D_{i,1}$→[t])

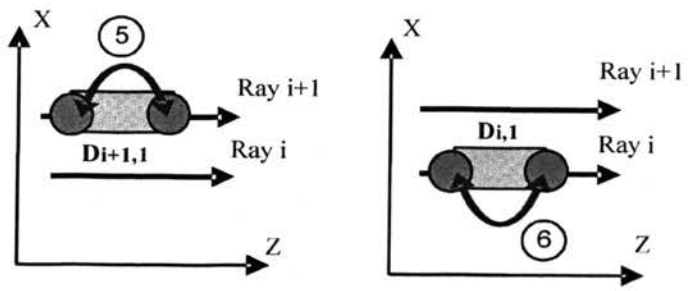


Figure 5. Special Cases of the Contouring Algorithm. (Left) When R_i=0 and (Right) When R_{i+1}=0

c) Step 3: Create a connection table to record the connections

After Step 2, each dexel point has exactly two connected dexel points, which are its adjacent vertices on the contour. In order to separate the points into different contours, a three-column connection table is created. The middle column lists the dexel points in the same sequence as they are generated and read. Their connecting points are stored in the left and right columns separately. In order to generate contours in the counterclockwise sequence, the left column is always filled first. After filling in all the connecting points in Step 2, as shown in Fig. 6, the table will be full without any empty spaces.

d) Step 4: Traverse the connection table to construct contours

The objective of the last step is to extract various contours from the connection table. The basic idea is to traverse the connecting points of one contour at a time, until all the contours have been extracted. The traversing sequence starts from the top to the bottom of the connection table. The starting point of a contour is the first unsearched

point. The next point of the contour is chosen based on one of two cases: in the *first case*, if none of its two connected points in the connection table has been searched, then the algorithm picks the one on the left as the next point on the contour; in the *second case* if the left point has been searched, then it takes the right point as the next point. This process continues until reaching a point (in the middle column) whose connected elements have been both searched. When this occurs, the contour is completed and the algorithm starts to search for another contour from the first unsearched point, if it exists, in the table. The search process continues until all the points have been traversed. The pseudo code is given in the appendix.

For example, Figure 6 starts from point 1 (p1). None of its two connected elements (p2 and p3) has been searched, so p2 is picked from the left column of p1. After checking p2, the unsearched point, p8, is picked from the right column of p2 because its left column has been searched. The rest of the points can be extracted in the same manner as listed in the sequence, p1→p2→p8→p14→p16→p15→p11→p5→p4→ p10→p9, until reaching p3. Both of the two connected points of p3 have been searched. Therefore, this contour is completed. Another contour begins from the first unsearched point, which is p6. The same procedure is repeated until all the points in the table have been searched. At the end, two contours are formed in the counterclockwise sequence on the right side of Fig. 6.

2.3. Contour Generation Example

A detailed example is given in Fig. 7 to illustrate the contour generation process following the above steps. Figure 7(a) is a slice of a 3D solid on the X-Z plane. The dexel data (b) are generated by the ray casting process. The bottom-right illustration shows the resulting one outer contour and three inner contours after all the dexel points have been connected.

2.4. Discussion of the Contour Generation Algorithm

The connection algorithm given in Section 2.2(b) requires the ray spacing to be small enough. Otherwise, problems may occur as illustrated in Fig. 8. The shapes of the slanted thin box and the slot in Fig. 8(a) can not be reconstructed, as shown in Fig. 8(b). Also, when two separate objects or holes are very close to each other in X-direction, as shown in (d), the reconstructed contour model in (e) has different topology from the

original shape in that the two neighboring objects (or holes) have been combined into one. The above problems can be solved by decreasing the distance between adjacent rays, thus, increasing the resolution of dexel data. For example, by decreasing the ray spacing, the generated contours in Figs. 8(c) and 8(f) have captured the original topologies of the models in (a) and (d), respectively.

Connected Point	Dexel Point	Connected Point
2	1	3
1	2	8
1	3	9
5	4	10
4	5	11
7	6	12
6	7	13
2	8	14
3	9	10
4	10	9
5	11	15
6	12	13
7	13	12
8	14	16
11	15	16
14	16	15

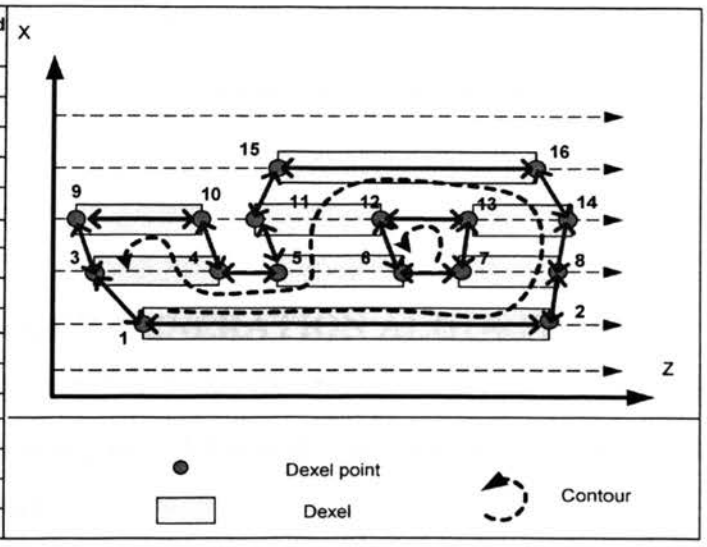


Figure 6. Traversing the Connection Table to Separate Contours

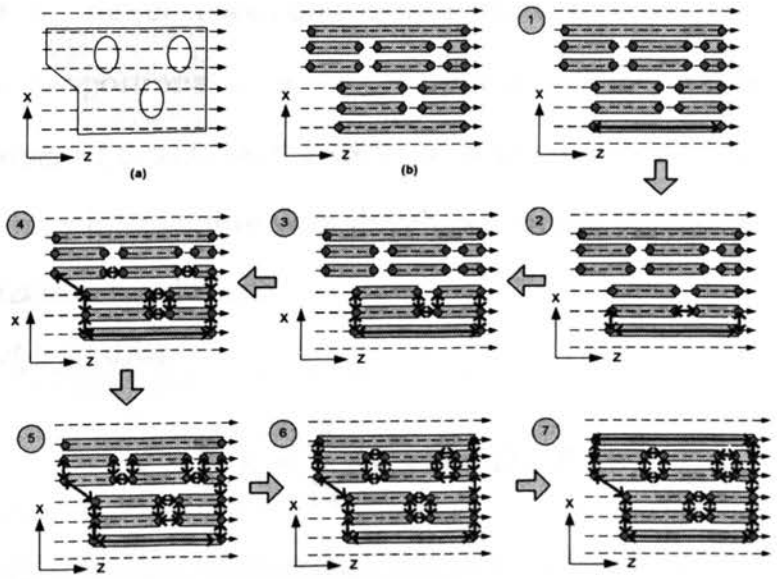


Figure 7. Example of the Contour Generation Process

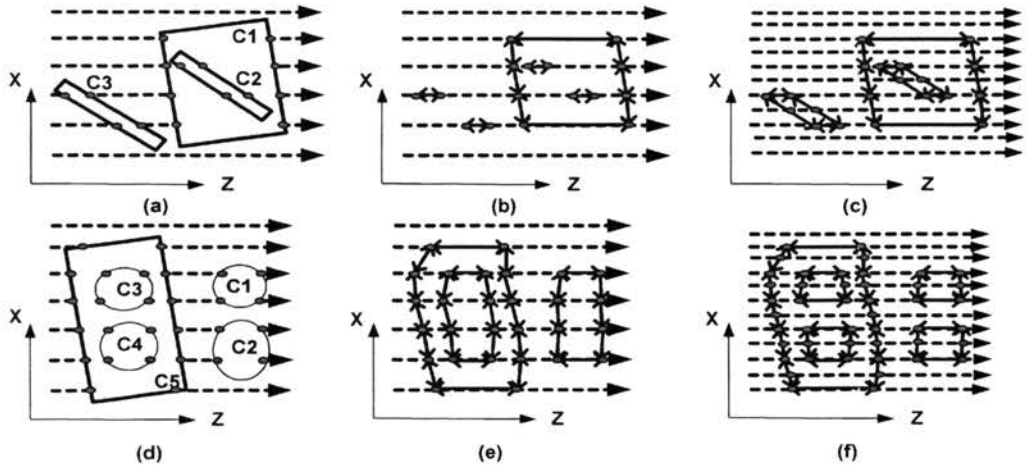


Figure 8. Discussion on the Validation of the Observations

3. ANALYSIS OF THE CONTOUR GENERATION ALGORITHM

In this section, the computational complexity and storage requirement of the contour generation algorithm are analyzed. Two test cases are used to verify the computational complexity analysis.

3.1. Computational Complexity Analysis

In Step 1 of the contour generation algorithm, the dexel spaces on Ray *i* are compared to the spaces on Ray *i*+1 to separate the dexels into groups. Given n_i dexel spaces on Ray *i* and n_{i+1} dexel spaces on Ray *i+1*, the computation time of comparing these dexel spaces is proportional to $n_i \times n_{i+1}$. Suppose *N* is the number of dexel spaces on a slice, *D* is the number of dexels on the slice, d_i is the number of dexels on Ray *i*, and β is the number of rays intersecting with the object for the considered slice, representing the model's discretization resolution.

Because $n_i = d_i + 1$, thus,

$$N = \sum_{i=1}^{\beta} n_i = \sum_{i=1}^{\beta} (d_i + 1) = D + \beta \tag{1}$$

where α designates the average number of dexels along a ray ($\alpha = \frac{D}{\beta}$), and thus, α represents the complexity of an object model. When $\beta >> d_i$, it is reasonable to assume that the average of $d_i d_{i+1}$ ($\overline{d_i d_{i+1}}$) is equal to the average of $d_i d_j$ ($\overline{d_i d_j}$), where i and j represent any two arbitrary rays. Then we have

$$D^2 = (\sum_{i=1}^{\beta} d_i)^2 = \sum_{i=1}^{\beta}\sum_{j=1}^{\beta} d_i d_j = \beta^2 \overline{d_i d_j} = \beta^2 \overline{d_i d_{i+1}} \tag{2}$$

$$\text{so} \quad \overline{d_i d_{i+1}} = \frac{D^2}{\beta^2} \tag{3}$$

The computation time, T_1, for Step 1 of the contour generation algorithm on one slice is

$$T_1 \propto \sum_{i=1}^{\beta-1}(n_i n_{i+1}) = \sum_{i=1}^{\beta-1}(d_i d_{i+1} + d_i + d_{i+1} + 1)$$

$$= (\beta-1)(\overline{d_i d_{i+1}} + \overline{d_i} + \overline{d_{i+1}} + 1)$$

$$= (\beta-1)(\frac{D^2}{\beta^2} + 2\frac{D}{\beta} + 1)$$

$$= (\beta-1)(\alpha+1)^2 \tag{4}$$

Thus, the computation time for Step 1 is $c_1(\beta-1)(\alpha+1)^2$, where c_1 is a constant. In Step 2 and Step 3 of the contour generation algorithm, the dexel points in each group are connected and the connection table is generated. These computation times are each proportional to $\alpha\beta$. In Step 4, the elements in the connection table are searched to obtain contours. The computation time is also proportional to $\alpha\beta$. Thus, the total computation time of the contour generation algorithm for one slice is:

$$T = c_1(\alpha+1)^2(\beta-1) + c_2\alpha\beta + c_3\alpha\beta + c_4\alpha\beta \tag{5}$$

By adding the computation times for the four steps, the computational complexity of the contour generation algorithm is $O(\alpha^2\beta)$ for each slice.

3.2. Memory Requirement Analysis

In the four-step contour generation algorithm, three sets of data are created to save (i) the initial dexel data, (ii) the connections between dexel points, and (iii) the final contour data. An *array* is used to store the initial dexel data which has the storage cost in

proportional to $\alpha\beta$. The connections between dexel points and the generated contours are also each saved in an *array* with the memory cost proportional to $\alpha\beta$. Overall, the memory requirement of the contour generation algorithm is proportional to the number of dexels on each slice.

3.3. Numerical Experiments

The implementation code for the contour generation and surface reconstruction is written in C++. It runs on a Microsoft Windows XP workstation with a 2.8G Hz CPU, 512 MB RAM, and a GeForce4 MX 420 graphics card with 64MB memory. The graphic-rendering component is developed with the OpenGL library.

Two numerical experiments were designed to verify the result of the above analyses. In the first test, the value of β is set constant, and the value of α is varied in the contour generation algorithm. As seen in Fig. 9, a series of skull models B_i are created along Z-axis, where i is the number of skull models. The same number of rays in Z direction is used in the ray casting process to generate dexel data so that the value of α is proportional to the number of skull models. The generated contour on every slice for B_1 is the same and is shown in Fig. 9(d).

Table 1 lists the computation time of the contour generation algorithm, the number of dexels on one slice for each of the skull models, and the value of α. β is the same (β=13,924) for all the skull models in this test. There is a quadratic relation between α and T as shown in Fig. 10. It is consistent with the results of analysis expressed by Eq. (5).

In the second test, for the same model, the value of β is changed. The single skull model (B_1) from the previous test is used. As shown in Fig. 11, a linear relation exists between the number of rays (β) and the computation time (T) of the contour generation algorithm. Table 2 lists the number of rays, the number of dexels, and the value of α as well as the computation time of the contour generation algorithm.

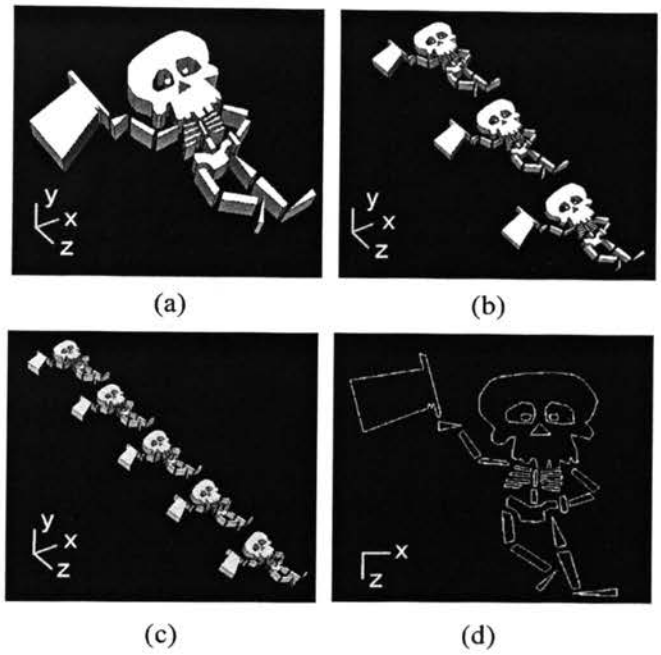


Figure 9. Numerical Experiments. (a)B_1, (b)B_3, (c)B_5, and the generated contour from B_1(d)

Table 1. Computation Results of the Contour Generation Algorithm

	B_1	B_2	B_3	B_4	B_5
No. of dexels (D)	45,666	91,332	136,998	182,664	228,330
Average no. of dexels per ray (α)	3.280	6.559	9.839	13.119	16.398
Contour generation time (sec)	0.161	0.386	0.700	1.116	1.687

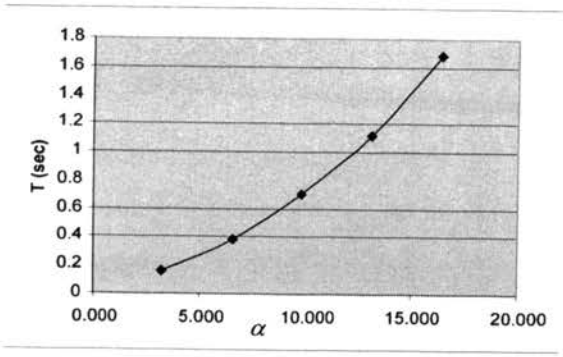


Figure 10. Contour Generation Time (T) vs. Average No. of Dexels Per Ray (α)

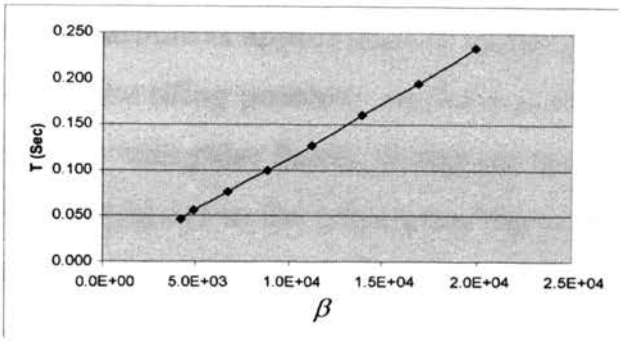


Figure 11. Contour Generation Time (T) vs. Number of Rays (β)

Table 2. Computation Time of the Contour Generation Algorithm

No. of rays (β)	No. of dexels (D)	Average no. of dexels/ray (α =D/ β)	Contour generation time (sec)
4,225	13,650	3.231	0.047
4,900	16,170	3.300	0.056
6,724	22,140	3.293	0.077
8,836	28,764	3.255	0.100
11,236	36,358	3.236	0.127
13,924	45,666	3.280	0.161
16,900	54,600	3.231	0.195
19,881	65,283	3.284	0.235

4. IMPLEMENTATION AND EXAMPLES

4.1. Surface Reconstruction by Tiling Contours

In order to finally generate the triangular facets of the object's boundary surface for viewing purpose, the algorithm developed by Christiansen and Sederberg [25] has been implemented to reconstruct the surface model from planar contours. This involves first solving the *correspondence* problem, i.e. determining which two contours on two adjacent slices are corresponding to each other. The overlapping ratio between the areas of two contours has been utilized, as described by Wang and Aggarwal [26], as the basis to choose corresponding contours. The overlapped area must exceed a certain value for two contours to be considered corresponding to each other. To speed up the computation, which is needed for applications such as virtual sculpting and NC machining simulation, the overlapping area of two contours is approximated by the overlapping area of their rectangular bounding boxes. The *tiling* problem, i.e. how to connect the vertices of two corresponded contours to form triangular facets, is tackled by connecting the points of one contour to their nearest neighbors in the corresponding contour after mapping the corresponding contour onto the same unit square, as described by Christiansen and Sederberg [25]. If one contour on a slice has correspondence to two or more contours on an adjacent slice, which is the so-called *branching* problem, a special step must be taken in tiling the corresponding contours. The branching problem is handled by connecting the closest points between two or more branched contours, so as to treat the multiple contours as one composite contour in the tiling process.

The above method works well when the two corresponding contours have similar shapes. However, if the shapes of two corresponding contours are very dissimilar, ambiguity becomes inevitable and some difficulty may occur. In that situation, the density of slices can be increased to reduce shape variations between corresponding contours on adjacent slices. After completing the process of corresponding, branching, and tiling, each tiled triangular facet consists of exactly one contour segment and two connection edges between the two corresponding contours.

4.2. Virtual Sculpting

The methods of contour generation and surface reconstruction presented above have been coded into computer software and incorporated in the development of an

experimental virtual sculpting system and an NC machining simulation system. Some of the research efforts on developing these systems have been discussed in previous papers [14-16, 27, 28].

A schematic of the virtual sculpting system is shown in Fig. 12. The goal of this experimental system is to provide the designer with an intuitive and interactive modeling environment including haptic interface capabilities such that the user can focus on the modeling intent. During the sculpting process, both the tool and the stock (initial workpiece) are modeled by boundary representation, where the object surface is a faceted approximation composed of connected, non-overlapping triangles. The sculpting tool is manipulated by the designer/stylist who holds and moves the stylus of the PHANToM^{TM} device. The tool position and orientation are tracked by the joint sensors in the PHANToM^{TM}. The swept volume formed by the movement of the tool between two consecutive sampling times is calculated using the sweep differential equation approach [29]. The workpiece and the tool swept volume are sent to the ray casting process to obtain their dexel representations. Boolean operations on dexels are obtained by comparing and merging the ranges of z-values of relevant dexels. The surface reconstruction module can be executed to convert the dexel model into a triangular mesh within seconds for displaying the sculpted model viewed from any directions. A multithread computation environment is built in our virtual sculpting system, which enables suitable update rates for the various components in the run time.

Figure 13 shows a freeform cat model created within the virtual sculpting system. The original cat model (without eyes in Fig. 13(a)) is imported into the system in the STL format. Eye cavities are first carved by sculpting the cat model with cylinder shaped cutters. Two eyeballs are then added and placed in the cavities by performing Boolean union with ball shaped cutters. The tail of the cat is also added. After applying the contour generation algorithm and tiling the generated contour into a triangular surface patch, the modified cat model can be viewed in any directions as shown in Fig. 13(c) and (d).

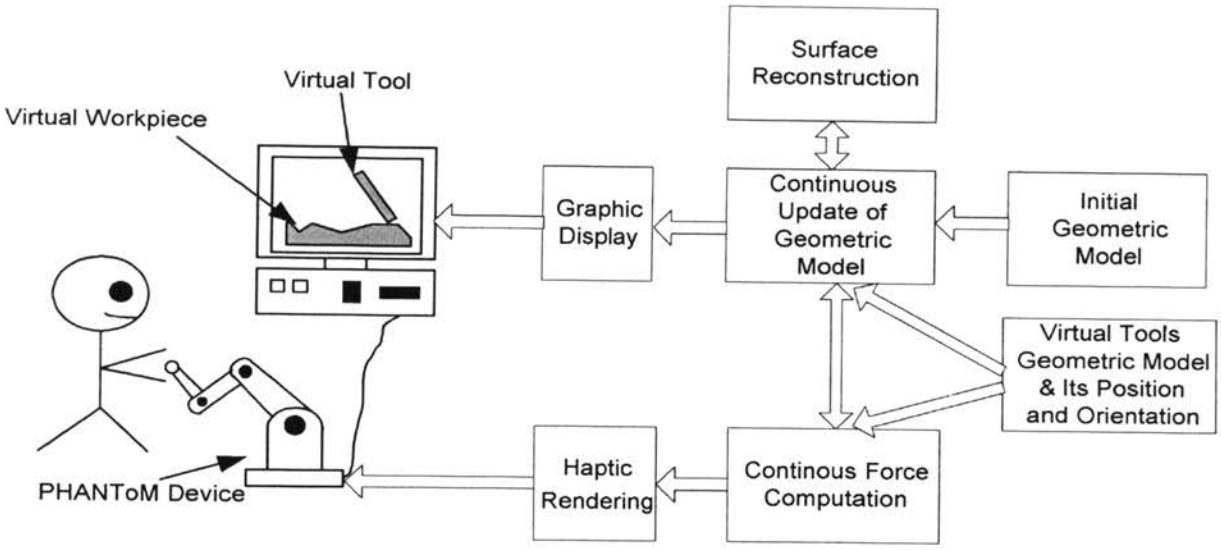


Figure 12. The Virtual Sculpting System Configuration

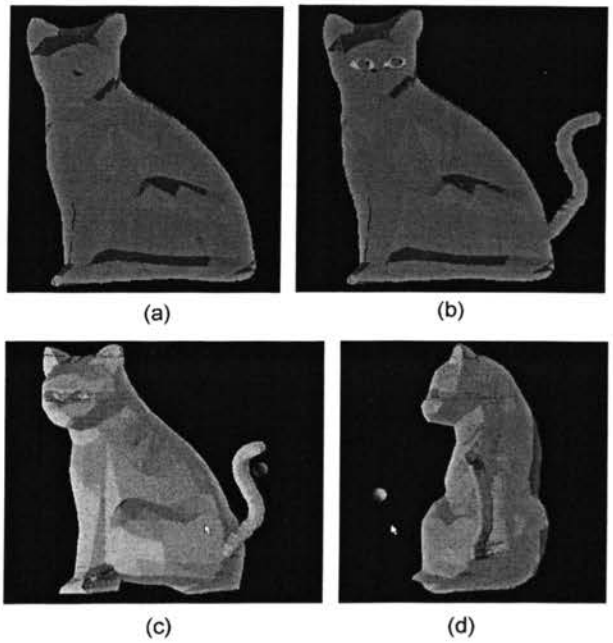


Figure 13. Modeling Examples. (a) The Imported Cat Model Created from A CAD System (b) Eyes and Tails Created by Virtual Sculpting (c) and (d) Viewing the Modified Cat Model in Different Directions

4.3. NC Machining Simulation

The NC machining simulation system has the same geometric modeling engine as the virtual sculpting system. The only difference is that in the NC machining simulation system, the cutter location file is generated by a CAD/CAM system, instead of by the designer/stylist in the case of virtual sculpting. Figure 14 shows a mouse model that is in the midst of NC machining simulation. It demonstrates that the triangular surface can be reconstructed from the dexel data interactively during the animation of simulated machining. The sculpted model can also be rotated in arbitrary angles to provide different views of the model.

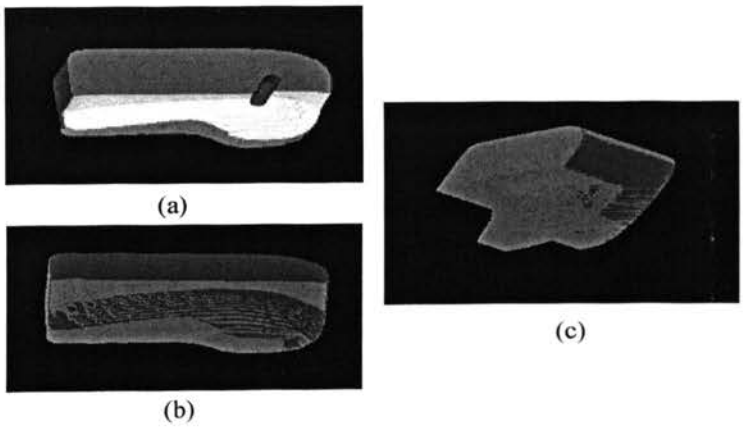


Figure 14. A Mouse in the Midst of NC Machining Simulation. (a), (b) and (c) Show the Generated Mouse Viewed from Two Different Directions after Performing Surface Reconstruction during the Machining Simulation

5. CONCLUSION

This paper has presented the development of a novel method to extract 2D contours from dexel data for the purpose of surface reconstruction for a 3D model. The surface reconstruction process solves the view-dependent problem inherent in dexel-based applications such as virtual sculpting and NC machining simulation. The dexel data are first put into different groups using a grouping criterion. Then the dexel points in the same group are connected using a set of connection rules. A connection table is created

which lists all the dexel points and their connected dexel points on one slice. Then the dexel points in the table are connected to construct the contours with points on each contour in a counterclockwise sequence. The generated contours are used to reconstruct the triangular surface model by implementing existing techniques, which are incorporated into our system. The computational complexity of the contour generation algorithm has been analyzed and verified with numerical experiments.

ACKNOWLEDGEMENTS

This research is supported by a National Science Foundation award (CCR-0310619) and a Ford Foundation grant, as well as by the Intelligent Systems Center at the Missouri University of Science and Technology.

REFERENCES

1. Ellis, J. L., Kedem, G., Lyerly, T. C., Thielman, D. G., Marisa, R. J., Menon, J. P., and Voelcker, H. B., 1991, "The RayCasting Engine and Ray Representations: A Technical Summary," International Journal of Computational Geometry and Applications, 1(4), pp. 347-380.
2. Menon, J., Marisa, R. J., and Zagajac, J., 1994, "More Powerful Solid Modeling Through Ray Representations," IEEE Computer Graphics and Applications, 14(3), pp. 22-35.
3. Fleisig, R. V., and Spence, A. D., 2005, "B-Rep Based Parallel Machining Simulation," Proc. of the 19th International Symposium on High Performance Computing Systems and Applications, IEEE Computer Society, pp. 83-89.
4. Chappel, I. T., 1983, "The Use of Vectors to Simulate Material Removed by Numerically Controlled Milling," Computer Aided Design, 15(3), pp. 156-158.
5. Maeng, S. R., Baek, N., Shin, S. Y., and Choi, K. Y., 2004, "A Fast NC Simulation Method for Circularly Moving Tools in the Z-Map Environment," Proc. of the Geometric Modeling and Processing, IEEE Computer Society, pp. 319-330.
6. Takafumi, S., and Tokiichiro, T., 1991, "NC machining with G-buffer Method," Proc. of the 18th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, pp. 207-216.

7. Puig, A., Perez-Vidal, L., and Tost, D., 2003, "3D Simulation of Tool Machining," Computers and Graphics, 27(1), pp. 99-106.
8. Jang, D., Kim, K., and Jung, J., 2000, "Voxel-Based Virtual Multi-Axis Machining," The International Journal of Advanced Manufacturing Technology, 16(10), pp. 709-713.
9. Erik, L.J.B., Nguyen, T.H.M., Ben, K., Peeraphan, N., Huang, R.Y., and Le, T. S., 2003, "The Stencil Buffer Sweep Plane Algorithm from 5-axis CNC Tool Path Verification," Computer Aided Design, 35(12), pp. 1129-1142.
10. Van Hook, T., 1986, "Real-Time Shaded NC Milling Display," Proc. of ACM SIGGRAPH, ACM Press, pp. 15-20.
11. Konig, A. H., and Groller, E., 1998, "Real-Time Simulation and Visualization of NC Milling Processes for Inhomogeneous Materials on Low-End Graphics Hardware," Proc. of the Computer Graphics International, IEEE Computer Society, pp. 338.
12. Muller, H., Surmann, T., Stautner, M., Albersmann, F., and Weinert, K., 2003, "Online Sculpting and Visualization of Multi-Dexel Volumes," Proc. of the 8th ACM Symposium on Solid Modeling and Applications, Seattle, Washington, ACM Press, pp. 258-261.
13. Ren, Y., Lai-Yuen, S. K., and Lee, Y-S., 2006, "Virtual Prototyping and Manufacturing Planning by Using Tri-Dexel Models and Haptic Force Feedback," Virtual and Physical Prototyping, 1(1), pp. 3-18.
14. Leu, M. C., Maiteh, B. Y., Blackmore, D., and Fu, L., 2001, "Creation of Freeform Solid Models in Virtual Reality," Annals of the CIRP, 50(1), pp. 73-76.
15. Peng, X., and Leu, M. C., 2004, "Interactive Solid Modeling in a Virtual Environment with Haptic Interface," in Virtual and Augmented Reality Applications in Manufacturing, Springer-Verlag Limited, Ong, S.K., and Nee, A.Y.C., eds., Springer, pp. 41-60.
16. Leu, M. C., Peng, X., and Zhang, W., 2005, "Surface Reconstruction for Interactive Modeling of Freeform Solids by Virtual Sculpting," Annals of the CIRP, 54(1), 2005.
17. Huang, Y., and Oliver, J. H., 1995, "Integrated Simulation, Error Assessment and Tool Path Correction for Five-Axis NC Milling," Journal of Manufacturing Systems, 14(5), pp. 331-344.

18. Zhu, W., and Lee, Y-S., 2005, "A Visibility Sphere Marching Algorithm of Constructing Polyhedral Models for Haptics Sculpting and Product Prototyping," International Journal of Robotics and Computer Integrated Manufacturing, 21(1), pp. 19-36.
19. Zhu, W., 2003, "Virtual Sculpting and Polyhedral Machining Planning System with Haptic Interface," Ph.D. thesis, North Carolina State University, Raleigh, NC, http://www.lib.ncsu.edu/theses/available/etd-08172003-194602/.
20. Lorensen, W. E., and Cline, H. E., 1987, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," Computer Graphics, 21(4), pp. 163-169.
21. Benouamer, M. O., and Michelucci, D., 1997, "Bridging the Gap between CSG and Brep via a Triple Ray Representation," Proc. 4th ACM Symposium on Solid Modeling and Applications, ACM Press, Atlanta, GA, pp. 68-79.
22. Edelsbrunner, H., Kirkpatrick, D.G., and Seidel, R., 1983, "On the Shape of a Set of Points in the Plane," IEEE Transactions on Information Theory, 29(4), pp. 551-559.
23. Kirkpatrick, D.G., and Radke J.D., 1985, "A Framework for Computational Morphology," in Computational Geometry, Toussaint, G. T., eds., North-Holland, pp. 217-248.
24. Veltkamp, R.C., 1992, "The γ-neighborhood Graph," Computational Geometry: Theory and Applications, 1(4), pp. 227-246.
25. Christiansen, H. N., and Sederberg, T. W., 1978, "Conversion of Complex Contour Line Definitions into Polygonal Element Mosaics," Proc. of the 5th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, pp. 187-192.
26. Wang, Y. F., and Aggarwal, J. K., 1986, "Surface Reconstruction and Representation of 3D Scenes," Pattern Recognition, 19(3), pp. 197-207.
27. Blackmore, D., Leu, M. C., and Wang, L., 1997, "The Sweep-Envelope Differential Equation Algorithm and Its Application to NC Machining Verification," Journal of Computer Aided Design, 29(9), pp. 629-637.
28. Maiteh, B. Y., Blackmore, D., Abdel-Malek, L., and Leu, M.C., 2000, "Swept-Volume Computation for Machining Simulation and Virtual Reality Application," Journal of Materials Processing and Manufacturing Science, 7, pp. 380-390.

29. Blackmore, D., and Leu, M. C., 1992, "Analysis of Swept Volume via Lie Groups and Differential Equations," International Journal of Robotics Research, 11(6), pp. 516-537.

II: SURFACE RECONSTRUCTION USING DEXEL DATA FROM THREE SETS OF ORTHOGONAL RAYS

Weihan Zhang and Ming C. Leu

Department of Mechanical and Aerospace Engineering
Missouri University of Science & Technology
Rolla, Missouri 65409, USA
Email: wzxq6@mst.edu, mleu@mst.edu

ABSTRACT

Triple-dexel modeling is a geometric representation method which depicts the intersection of a solid with rays cast in three orthogonal directions. Due to its fast Boolean operations, simple data structure and easy implementation, triple-dexel modeling is highly suitable for real-time graphics-based simulation applications such as NC machining verification and virtual sculpting. This paper presents a novel surface reconstruction method from triple-dexel data by first converting the triple-dexel data into contours on three sets of orthogonal slices and then generating the solid's boundary surface in triangular facets from these contours. The developed method is faster than the voxel-based method, and the reconstructed surface model is more accurate than the surface reconstructed from voxel representation using the marching cube algorithm. Examples are given to demonstrate the ability of surface reconstruction from the triple-dexel model in virtual sculpting.

1. INTRODUCTION

Van Hook [1986] introduced the notion of dexel as an abbreviation for "depth element." Single-dexel representation of a solid, also called ray representation, is constructed via a process of computing intersections between the solid and rays cast in one direction. For a given solid, a set of parallel and equidistant rays are projected and intersected with the solid. For each ray the intersected points are stored in the following manner. First, a dexel is defined by two intersection points in a line segment that is

completely inside the solid. Then the dexels on a ray are sorted and concatenated into a linked list structure. Finally, the dexel lists are organized into a dexel matrix, which represents the single-dexel model as shown in Fig. 1. Single-dexel modeling is among the most notable approximation methods used to support NC machining simulation [Huang and Oliver, 1995; Konig and Groller, 1998] and virtual sculpting [Peng and Leu, 2003; Leu et al., 2005; Peng et al., 2006] because it allows fast and robust Boolean operations, needs little memory, and has simple data structures for real-time simulation. However, in the single-dexel model, low sampling quality occurs in regions where the surface normals are nearly perpendicular to the ray direction. To address this problem, a triple-dexel model can be constructed by casting rays in three orthogonal directions (normally in x, y, and z directions) to discretize the model, as shown in Fig. 2. This model is also used in NC machining simulation [Muller et al., 2003] and virtual sculpting [Ren et al., 2006].

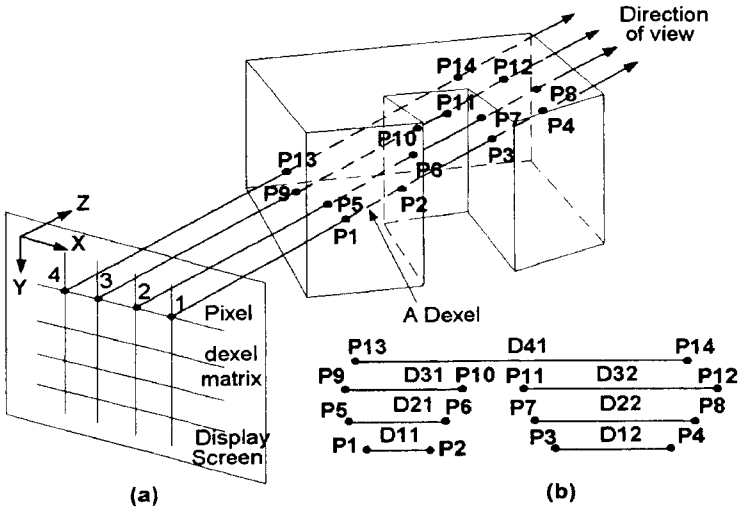


Figure 1. Illustration of the Ray-Casting Process and the Single-Dexel Representation

The conversion from the triple-dexel data of a 3D model into triangular surface patches is an important issue. The reconstructed triangular facets can be used by conventional CAD/CAM/CAE systems to perform geometric design, engineering

analysis, and automated manufacturing applications. Further, the triangulated 3D model can be viewed in any directions as desired using standard routines of computer graphics software. The surface reconstruction from triple-dexel is also difficult because reconstruction methods have to overcome topological ambiguity, which is usually being dealt through grid based methods.

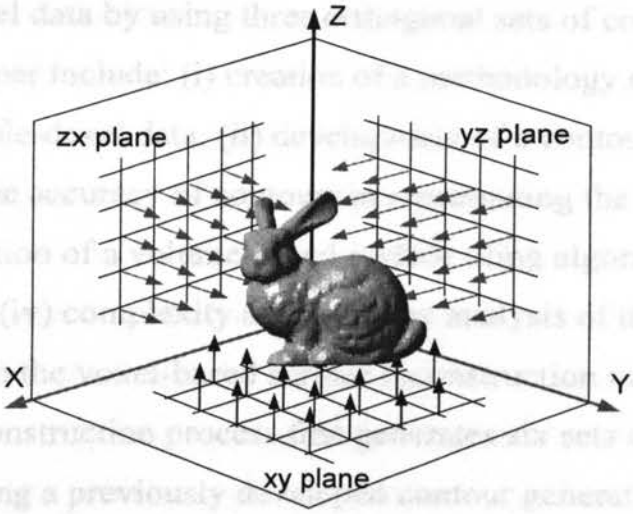


Figure 2. Construction of a Triple-Dexel Model

Benouamer and Michelucci [1997] utilized the marching cube algorithm to generate the triangular surface model from triple-dexel data by first generating voxel data from the triple-dexel data. However, the reconstructed surface may suffer from topology errors and poor approximation of sharp features. In addition, the computations are expensive because the computation complexity is proportional to the number of voxels. In a sense the triple-dexel data can be converted into point clouds and reconstructed using Delaunay triangulation or surface fitting methods available from the existing literature. However, the Delaunay triangulation and surface fitting processes are also computationally expensive. Triple-dexel data can be also converted into parallel slices where triangular surfaces can be reconstructed using surface tilling algorithms [Barequet

and Vaxman, 2007]. However, the topology issue is still a major problem for the surface reconstruction. Svitak and Skala [2004] have shown that contours on three orthogonal slices offer connectivity information among dexel points on each slice, thus, not only the reconstructed surface model is topological correct and accurate, but also the reconstruct process is fast. However, to our best knowledge, there has been no previous work on generating contours on three sets of orthogonal slices from triple-dexel data for the purpose of reconstructing a surface model.

The work described in the present paper is the first to reconstruct a triangular surface from triple-dexel data by using three orthogonal sets of contours. Our main contributions in this paper include: (i) creation of a methodology of surface reconstruction from triple-dexel data, (ii) development of a contour combination algorithm to improve the accuracy of contours in representing the 3D model's cross sections, (iii) incorporation of a volume-based surface tiling algorithm in the surface reconstruction process, (iv) complexity and accuracy analysis of the developed method, and (v) comparison with the voxel-based surface reconstruction method.

Our surface reconstruction process first generates six sets of planar contours from the triple-dexel data using a previously developed contour generation algorithm [Zhang et al., 2007]. A contour combination algorithm developed in the present paper is then used to combine two sets of corresponding contours on parallel slices into one set of contours along each of x, y and z axes, forming a total of three sets of contours on slices parallel to xy, yz, and zx planes. A volume-based surface tiling algorithm [Svitak and Skala, 2004] is then utilized to generate triangular facets for the boundary surface of the concerned 3D model from these three sets of contours. Then the developed method is analyzed in terms of computational complexity, memory cost, and accuracy of the reconstructed surface. Numerical experiments show that the developed method generates a more accurate surface than that reconstructed from voxel data under the same grid resolution and the level of the object's details, and that this method is more efficient than the voxel-based method.

The paper is structured as follows. In Section 2, related work on triple-dexel based modeling and surface reconstruction methods is reviewed. Section 3 details our method of surface reconstruction including how to correspond and combine the generated

contours, and how to tile the three orthogonal sets of contours into triangular facets. Computational complexity, storage requirement, and surface errors are analyzed in Section 4. Section 5 describes implementation examples of the developed surface reconstruction process and applications in virtual sculpting. The triple-dexel modeling is compared with the voxel representation for surface reconstruction in Section 6. Conclusions are drawn in Section 7.

2. RELATED WORK

2.1. Triple-Dexel Based Solid Modeling

Triple-dexel modeling is an extension of single-dexel modeling for the purpose of improving data sampling quality. The memory cost of a triple-dexel model is proportional to the surface area of the solid model and the ray density (no. of rays per unit area), and the time of accessing the linked list structure that stores the dexel data is proportional to the number of dexel elements in the list. Muller et al. [2003] developed a triple-dexel based online milling simulation system and Ren et al. [2006] developed a virtual sculpting system with haptic feedback by using the triple-dexel model. However, both of these studies did not reconstruct triangularized surface models, which are very useful for visualization and other purposes in CAD/CAM/CAE.

2.2. Surface Reconstruction from Triple-Dexel Data

Dexel modeling has a view-dependent problem because the ray directions are fixed and the dexel data only records the geometric information of a 3D object in the ray direction(s), as seen in Figs. 1 and 2. Thus, in the practice of dexel-based simulation without surface reconstruction, only a limited number of views can be generated for the simulation, unlike the generation of a surface model which can be viewed from any desired direction. To solve the view-dependent problem for triple-dexel data, Benouamer and Michelucci [1997] applied the marching cube algorithm [Lorensen and Cline, 1987] to generate the 3D object's boundary surface. Although simple and powerful, this technique suffers from poor approximation of sharp features and may encounter ambiguous cases in the surface reconstruction process. Muller et al. [2003] implemented the point-based rendering method developed by Pfister et al. [2000] for their online

sculpting system. However, it was difficult to interface the sculpted models with CAD/CAM/CAE systems for further design and analysis.

Another related research is the study of surface reconstruction from point clouds since dexel data can be treated as point cloud data in 3D space. Literature in this research comes mainly from the fields of image processing, computational geometry and computer graphics [Azernikov et al., 2003]. Delaunay-based methods [Edelsbrunner and Mucke, 1994; Bernardini et al., 1999; Amenta et al., 2001; Dey et al., 2001] have been shown successful to produce a triangular mesh from point cloud data. However, the ball-pivoting algorithm [Bernardini et al., 1999] took 2.1 minutes to reconstruct 361K samples on 450MHz Pentium II Xeon PC, and the power crust method [Amenta et al., 2001] took about 6 minutes to reconstruct 30,000 samples on a 400 MHz Sun computer. Besides Delaunay-based methods, surface fitting techniques [Carr et al., 2001; Alexa et al., 2001; Ohtaka et al., 2003, 2006] have become popular recently for surface reconstruction because of their ability to account for noise in the input data. Nevertheless, one of the fastest implicit surface fitting methods [Ohtaka et al. 2006] still took 42 seconds to reconstruct the surface from a 362K input data on a 1.6 GHz Pentium IV PC. Our contour based method developed in this paper for surface reconstruction from triple-dexel data is more than one order of magnitude faster than the Delaunay triangulation or surface fitting based methods.

3. SURFACE RECONSTRUCTION FROM TRIPLE-DEXEL DATA

The main idea of the proposed surface reconstruction method is to generate contours from triple-dexel data on three sets of orthogonal slices, and utilize these contours to reconstruct the boundary surface of the 3D model. Overall, the method has three main steps. First, the contour generation algorithm takes the dexel data in each of x, y, and z directions as the input and generates planar contours on two orthogonal sets of parallel slices. For example, the dexel data in x direction is used to generate xy contours and xz contours. Next, on each set of parallel slices, the two sets of contours generated from the first step are combined into one set of contours. For example, an xy contour is combined with a yx contour on the same slice to generate a contour parallel to xy plane. After these two steps, there are three sets of contours (i.e., contours on planes parallel to

xy, *yz* and *zx* planes). In the last step, a volume-based tiling algorithm is utilized to generate triangular facets of the solid's boundary surface from the three sets of contours. The schematic diagram of the proposed method is shown in Fig. 3.

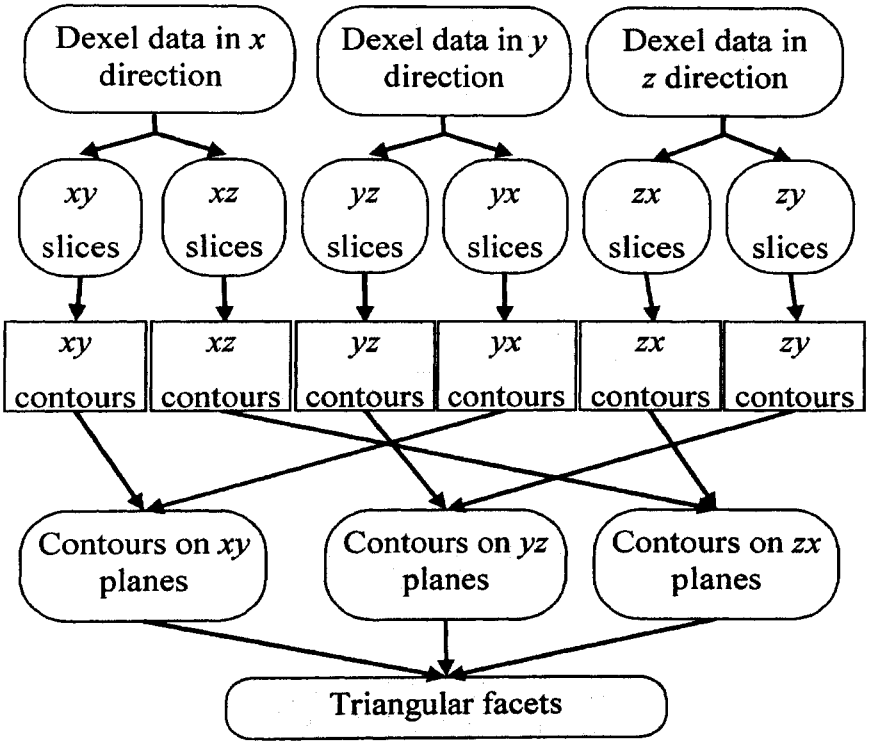


Figure 3. Proposed Method of Surface Reconstruction from Triple-Dexel Data

3.1. Contours Generation Algorithm

A contour generation algorithm has been developed to reconstruct contours from single-dexel data [Zhang et al., 2007]. A main difficulty in developing such an algorithm is to identify the dexel points which lie on inner contours, such as point 6 and point 12 in Fig. 4, and to construct connections between the dexel points. Central to the developed algorithm is a grouping criterion, which categorizes the dexels on two adjacent rays into different groups. The main idea of the grouping criterion is realizing that two overlapping dexel spaces, such as the space between points 12 and 13 and the space between points 6

and 7 in Fig. 4, on two adjacent rays may form part of an inner contour, such as the connections between points 6, 7, 13 and 12. Thus, the end points of the two overlapped dexel spaces are connected, e.g. point 6 is connected with point 12 and point 7 is connected with point 13. Overlapped dexel spaces on adjacent rays separate dexels into different groups, and within each group the end points of the dexel spaces are connected. Overall, this contour generation algorithm has four steps: first, dexels on every two adjacent rays are categorized into groups according to the grouping criterion. Second, inside each group, adjacent dexel points are connected. Third, a connection table is created to record connections between dexel points. Finally, the connection table is traversed to construct contours in a counterclockwise sequence.

Connected Point	Dexel Point	Connected Point
2	**1**	3
1	**2**	8
1	**3**	9
5	**4**	10
4	**5**	11
7	**6**	12
6	**7**	13
2	**8**	14
3	**9**	10
4	**10**	9
5	**11**	15
6	**12**	13
7	**13**	12
8	**14**	16
11	**15**	16
14	**16**	15

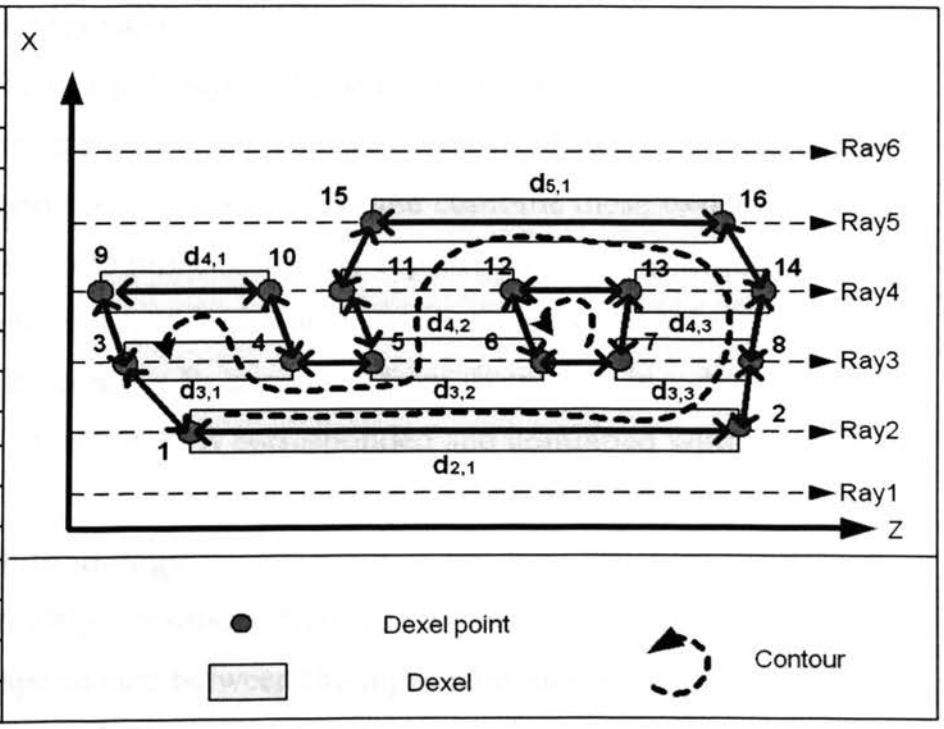


Figure 4. Contour Generation from Single-Dexel Data

To illustrate the contour generation process, Fig. 4 is used as an example. On Ray 4 and Ray 5, dexels $d_{4,2}$, $d_{4,3}$, and $d_{5,1}$ are in one group because they have overlaps, and $d_{4,1}$ is in another group. Thus, points 12 and 13, points 11 and 15, and point 14 and point 16 are connected. Because dexel $d_{4,1}$ is a top dexel, points 9 and 10 are connected. Once all the connections are made for every two adjacent rays, a connection table can be created and all the connections are listed in the table as shown on the left side of Fig. 4, where the middle column lists the dexel points in the same sequence as they are generated and read. Their connecting points are stored in the left and right columns separately. In order to generate contours in the counterclockwise direction, the left column is always filled with the smaller index. Finally, the sequence of points for each contour is generated by following the connection from one point to the next, until eventually coming back to the first point. The details of this algorithm are described in [Zhang et al., 2007].

3.2. Contour Combination Algorithm

After applying the contour generation algorithm to triple-dexel data, two sets of contours are present on planes parallel to each of *xy*, *yz* and *zx* planes. The objective of the contour combination algorithm is to correspond and combine these two sets of contours into one set of contours to more accurately represent the cross-sectional profiles of the 3D model. For example, in Fig. 5, contour A_1 generated from dexel data in *x* direction is corresponded with contour B_1 generated from dexel data in *y* direction to create contour C_1. Likewise, contour A_2 is corresponded and combined with B_2 to generate contour C_2.

3.2.1. Algorithm Design Methodology

Since there may be multiple contours A_i (i=1,...,u) and B_j (j=1,..., u) on each slice, the first step is to find correspondence between the input contours. Then, for each two corresponded contours (i.e., A_i and B_j), a starting pair of points from A_i and their associated points from B_j are found. After inserting these associated points from B_j into A_i, we continuously search for the rest of point pairs from A_i, which have at least one associated point from B_j in between, and insert the associated points. Finally, the combined contour is created when all the points from B_j have been inserted into A_i.

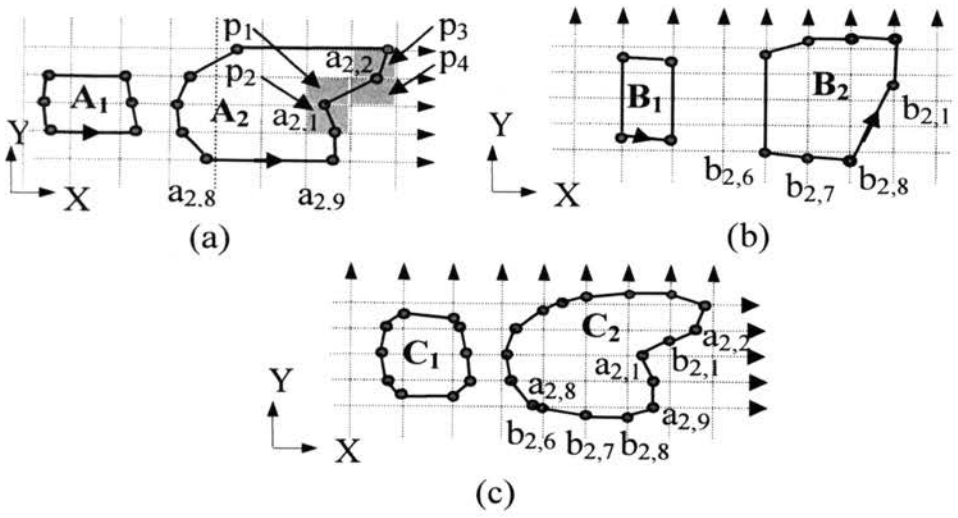


Figure 5. Contour Combination Algorithm. (a) xy Contours, (b) yx Contours and (c) the Combined Contours

Here a local connection method is developed by realizing both input contours are in the counterclockwise direction. If we scan the points of contour A_i in the counterclockwise sequence, the points of contour B_j should be continuously added to contour A_i in the same counterclockwise sequence. For example, a sequence of three points $b_{2,6}$, $b_{2,7}$, and $b_{2,8}$ are added between points $a_{2,8}$ and $a_{2,9}$ in Fig. 5(c). This implies that if the first ($b_{j,f}$) and the last ($b_{j,l}$) associated points between points $a_{i,k}$ and $a_{i,k+1}$ are correctly selected, it is trivial to find the rest of the associated points in between and insert them into A_i. It also implies that point $b_{j,l+1}$ is the next point to be considered for inserting into contour A_i. So it is only necessary to check the next pair of points of contour A_i to see if $b_{j,l+1}$ is in between. If so, $b_{j,l+1}$ and its following points from contour B_j can be added until the last associated point ($b_{j,o}$) is identified according to the criteria defined in Section 3.2.3. This process is repeated until all the points from contour B_j have been added to contour A_i. This contour combination method is efficient because of using the point sequence information in the input contours.

3.2.2. Contour Correspondence

The contour correspondence problem involves finding which contour from contour set A is to be combined with which contour from contour set B. The overlapping

area ratio [Wang and Aggarwal, 1986] between two contours has been utilized as the criterion to deal with this correspondence problem. The overlapping ratios between A_i and all the contours from contour set B are firstly calculated. Then the contour which has the maximum overlapping ratio with A_i is chosen. Likewise, every other contour in contour set A can be corresponded with a contour in contour set B. Here the numbers of input contours from each set are assumed to be equal. To speed up the calculation, the overlapping area between contour A_i and contour B_j is approximated by the overlapping area of their bounding boxes.

3.2.3. Contour Combination

The contour combination algorithm consists of two main steps to combine the corresponded contours (say, A_i and B_j). The first step is to identify the starting pair of points $a_{i,k}$ and $a_{i,k+1}$ of contour A_i, to find their associated points (i.e., $b_{j,f}, \ldots, b_{j,l}$) and to add them between $a_{i,k}$ and $a_{i,k+1}$. The second step is to continuously search from $a_{i,k+1}$ and $a_{i,k+2}$ to find the next pair of points in contour A_i which has at least one associated point from contour B_j. Then the associated points are identified starting from $b_{j,l+1}$ and onwards in B_j for insertion. The second step is repeated until all the points from contour B_j have been added to contour A_i. The details of this two-step algorithm are described below.

Step 1: Searching the starting pair of points from A_i and their associated points from B_j

Suppose contour A_i is generated from rays in x direction. two adjacent points $a_{i,k}$ and $a_{i,k+1}$ are firstly identified in A_i such that there is at least one y-directional ray in between. Mathematically, this requires

$$\mathrm{INT}[(a_{i,k} \rightarrow [x])/\Delta x] \neq \mathrm{INT}[(a_{i,k+1} \rightarrow [x])/\Delta x] \qquad (1)$$

where INT is a function to remove the decimal part of a number and return the resulting integer, Δx is the grid length in x direction, and $a_{i,k} \rightarrow [x]$ is the x value of point $a_{i,k}$. The pixels containing the first pair of points $a_{i,k}$ and $a_{i,k+1}$ are required to have *at most one ray intersecting point appearing on any edge of these pixels*. If $a_{i,k}$ and $a_{i,k+1}$ are on the same ray as shown in Fig. 6(a), pixels p_1, p_2, p_3 and p_4 must satisfy this requirement. If $a_{i,k}$ and $a_{i,k+1}$ are on two adjacent rays as shown in Fig. 6(b), pixels p_5, p_6, p_7 and p_8 which are the pixels containing $a_{i,k}$ and $a_{i,k+1}$, must satisfy this requirement. Taking contour A_2 in Fig. 5(a) as an example, points $a_{2,1}$ and $a_{2,2}$ can be the starting pair because there is a y ray

between them, and that the pixels p_1, p_2, p_3 and p_4 have no more than one ray intersecting point on any of their edges. Note that there could be many pairs of $a_{i,k}$ and $a_{i,k+1}$ that satisfy the requirements. Any of these pairs can be used as the starting pair. However, it is possible that the first pair of points is not found because of an insufficient number of rays. In such case, we can always cast additional rays to increase the ray density as discussed in Sec. 3.2.4.

Without loss of generality, in the starting pair of points, $a_{i,k}$ is assumed on the left side of $a_{i,k+1}$. Then the first associated point ($b_{j,f}$) and the last associated point ($b_{j,l}$) from B_j for points $a_{i,k}$ and $a_{i,k+1}$ can be found as follows: $b_{j,f}$ is on the line segment $l_{i,f}$, which is on the y ray immediately to the right of $a_{i,k}$; and $b_{j,l}$ is on the line segment $l_{i,l}$, which is on the y ray immediately to the left of $a_{i,k+1}$. As shown in Fig. 6(a), if $a_{i,k}$ and $a_{i,k+1}$ are on the same ray, each of $l_{i,f}$ and $l_{i,l}$ consists of two pixel edges. If $a_{i,k}$ and $a_{i,k+1}$ are on two adjacent rays, each of $l_{i,f}$ and $l_{i,l}$ consists of one pixel edge as shown in Fig. 6(b). In this case, the points on $l_{i,f}$ and $l_{i,l}$ in (b) are the first and the last associated points, respectively, from contour B_j.

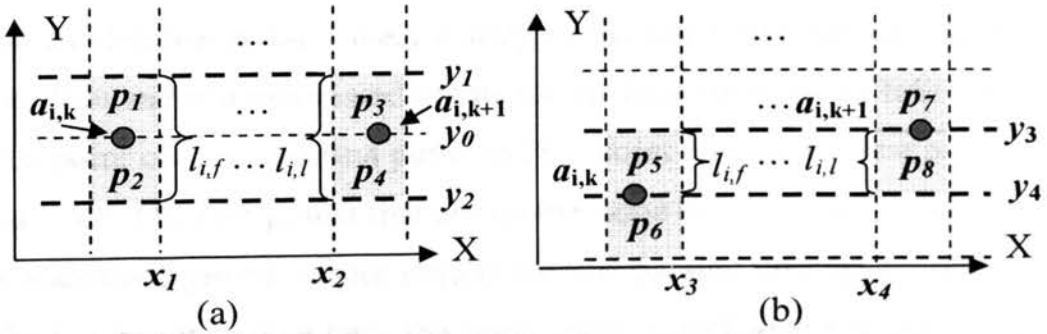


Figure 6. Locations of the First and the Last Associated Points of Contour B_j. (a) $A_{i,K}$ and $A_{i,K+1}$ on the Same Ray and (b) $A_{i,K}$ and $A_{i,K+1}$ on Adjacent Rays

To pinpoint the exact first and last associated points from contour B_j when $a_{i,k}$ and $a_{i,k+1}$ are on the same ray as shown in Fig. 6(a), Table 1 is created with four possible

combinations of their locations. For example, CA1 in Table 1 represents the case of having one point on $l_{i,f}$ and one point on $l_{i,l}$, and CA2 represents the case of having one point on $l_{i,f}$ and two points on $l_{i,l}$. Note that on each of $l_{i,f}$ and $l_{i,l}$, there is at least one point from B_j because the contour is continuous.

Table 1. Combinations of the First and the Last Associated Points

No. of points on $l_{i,l}$ \ No. of points on $l_{i,f}$	1	2
1	CA1	CA2
2	CA3	CA4

For each of the four cases, the first and last associated points from contour B_j are determined based on the reasoning that the line segment connecting these two points has no intersection with any rays parallel to x axis, otherwise points $a_{i,k}$ and $a_{i,k+1}$ would not have been two adjacent points of contour A_i. Note that the first point and the last point would be an identical point if there is only one y-directional ray passing between $a_{i,k}$ and $a_{i,k+1}$. The first and last associated points for the four cases are as follows:

CA1: The point on $l_{i,f}$ is the first point and the point on $l_{i,l}$ is the last point.

CA2 and CA3: The two points that are on the same side of $y=y_0$ in Fig. 6(a) are the first and last associated points, where $y=y_0$ is the line passing through points $a_{i,k}$ and $a_{i,k+1}$.

CA4: The two points which have the same counterclockwise sequence as points $a_{i,k}$ and $a_{i,k+1}$ are the first and last associated points. For example, in Fig. 7, between the two adjacent points $a_{i,k}$ and $a_{i,k+1}$, there are two sets of points $b_{j,g} \rightarrow b_{j,g+2}$ and $b_{j,r+2} \rightarrow b_{j,r}$ as the candidate associated points from B_j. Because $b_{j,g}$ and $b_{j,g+2}$ have the same directional sequence as points $a_{i,k}$ and $a_{i,k+1}$, they are the first and last associated points. The pseudo code for Step 1 is given in the Appendix.

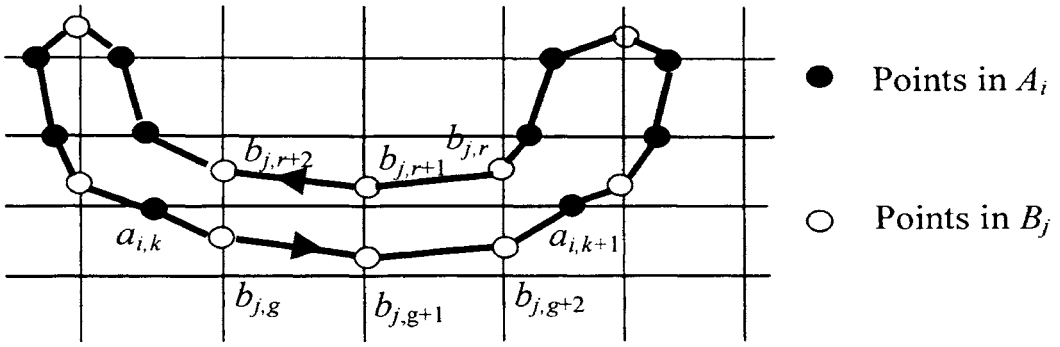


Figure 7. Illustration of the Solution to the Case CA4

Step 2: Continuously searching for the next pair of points from A_i and their associated points from B_j

From the result of Step 1, $b_{j,l+1}$ is the next candidate point to be added to contour A_i, so we only need to continuously search contour A_i from $a_{i,k+1}$ and $a_{i,k+2}$ to find the next pair of points $a_{i,r}$ and $a_{i,r+1}$ to consider inserting $b_{j,l+1}$, and to find the last associated point from contour B_j between points $a_{i,r}$ and $a_{i,r+1}$.

According to Eq. (1), for two adjacent points $a_{i,r}$ and $a_{i,r+1}$, if

$$\mathrm{INT}[(a_{i,r} \to [x])/\Delta x] \neq \mathrm{INT}[(a_{i,r+1} \to [x])/\Delta x] \tag{2}$$

then $a_{i,r}$ and $a_{i,r+1}$ must have associated points from contour B_j in between. The first one is $b_{j,l+1}$. To find the last associated point ($b_{j,o}$) between $a_{i,r}$ and $a_{i,r+1}$, we can search contour B_j from $b_{j,l+1}$ onward to find the point, $b_{j,o}$, that satisfies the following criteria:

$b_{j,o}$ is on one of the edges of the pixels that contain $a_{i,r+1}$

the y value of $a_{i,r+1}$ is between the y values of $b_{j,o}$ and $b_{j,o+1}$.

Mathematically, this requires

$$a_{i,r+1} \to [y] \in [\min(b_{j,o} \to [y], b_{j,o+1} \to [y]),\quad \max(b_{j,o} \to [y], b_{j,o+1} \to [y])] \tag{3}$$

where $a_{i,r+1} \to [y]$ represents the y value of point $a_{i,r+1}$. For example, Figure 8(a) shows two points $a_{i,r}$ and $a_{i,r+1}$ from A_i satisfying Eq. (2). Since we know $b_{j,z}$ is the first associated point and because $b_{j,z+2}$ satisfies the above criteria, $b_{j,z+2}$ is the last associated

point between points $a_{i,r}$ and $a_{i,r+1}$. After inserting the points, the combined contour of contours A_i and B_j is shown in Fig. 8(b).

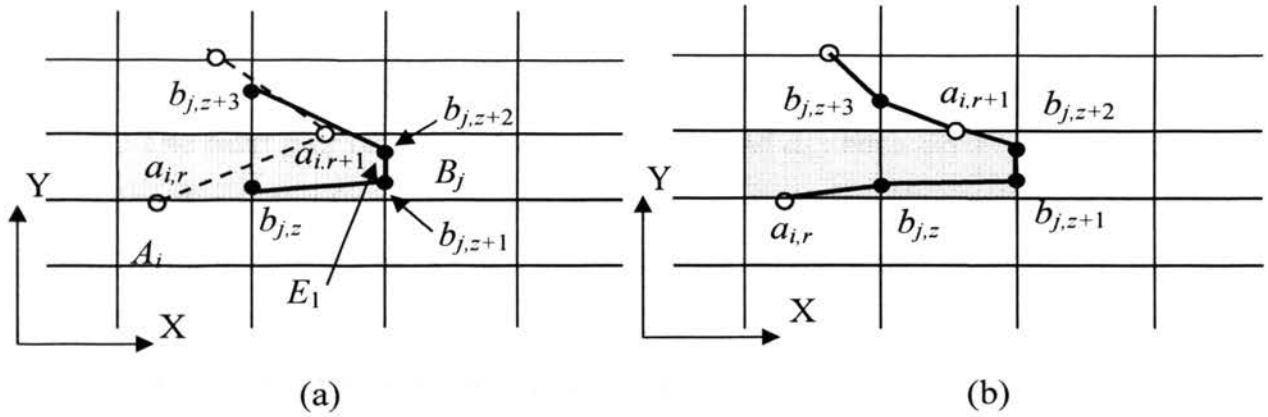


Figure 8. Contour Combination Process. (a) Two Generated Contours from the Contour Generation Process and (b) the Combined Contour

If two adjacent points $a_{i,r}$ and $a_{i,r+1}$ satisfy the following relationship

$$\mathrm{INT}[(a_{i,r} \rightarrow [x])/\Delta x] = \mathrm{INT}[(a_{i,r+1} \rightarrow [x])/\Delta x] \tag{4}$$

there is still a possibility that $a_{i,r}$ and $a_{i,r+1}$ have associated points from B_j in between. If $b_{j,l+1}$ satisfies the following criteria:

$b_{j,l+1}$ is on one of the edges of the pixel that contains both $a_{i,r}$ and $a_{i,r+1}$

the y value of $a_{i,r}$ is between the y values of $b_{j,l}$ and $b_{j,l+1}$

then $a_{i,r}$ and $a_{i,r+1}$ will have at least one associated point, and the first one is $b_{j,l+1}$. In this case, the same criteria as those given above can be utilized to find the last associated point ($b_{j,o}$) in between. If $a_{i,r}$ and $a_{i,r+1}$ have no associated point from B_j in between, the next pair of points in A_i, i.e., $a_{i,r+1}$ and $a_{i,r+2}$ will be checked to see if they have any associated point in between according to the above criteria. By repeating Step 2 for the rest of points in A_i until all the points in contour B_j have been added to contour A_i, the combined contour is finally obtained. The pseudo code for *Step 2* is given in the Appendix.

Figure 9 is taking as an example to illustrate the contour combination algorithm. The original contour in Fig. 9(a) is sampled by dexel data in x direction and y direction. The generated contours are shown in Fig. 9(b) as contour A_i and B_j. Contour A_i is searched to find a starting pair of points, $a_{i,1}$ and $a_{i,2}$, which satisfy the criterion of having at most one ray intersecting point on any edge of the pixels (p_a, p_b, p_c and p_d in Fig. 9(a)) containing $a_{i,1}$ or $a_{i,2}$.

The first associated point can be easily determined as point $b_{j,1}$ by searching contour B_j. The next pair of points from A_i are $a_{i,2}$ and $a_{i,3}$ because their y values satisfy Eq. (2). Thus, $b_{j,2}$ is the first associated point. $b_{j,2}$ is also tested to see if it is the last associated point according to the two given criteria. Because $b_{j,2}$ is on one of the edges of the pixel that contains $a_{i,3}$, and the y value of $a_{i,3}$ is between the y values of $b_{j,2}$ and $b_{j,3}$, thus, $b_{j,2}$ is the last and the only associated point between $a_{i,2}$ and $a_{i,3}$. Similarly, the associated points between $a_{i,3}$ and $a_{i,4}$ can be found to be points $b_{j,3}$, $b_{j,4}$, $b_{j,5}$, and $b_{j,6}$. By repeating the same procedure for the rest of points until every point in contour B_j has been added into A_i, the combined contour is generated as shown in Fig. 9(c).

3.2.4. Discussion

In the contour corresponding process, the numbers of input contours from the two contour sets are assumed to be equal. However, if the distance between adjacent rays is not small enough, the reconstructed contours from dexel data in two orthogonal directions may have different topologies. For example, in Fig. 10(a), the input contour is sampled with x directional and y directional rays. The generated contours are shown in (b) and (c) of the same figure. Because the small edge between points PT_1 and PT_2 is sampled by a ray in x direction, but not by any ray in y direction, one contour is generated from the x-dexel data but two contours are generated from the y-dexel data. In this case, the contour combination algorithm would fail. This problem can be solved by increasing the ray resolution in the contour generation process until the generated contours from dexel data in two orthogonal directions have the same topology. For example, the same contour of Fig. 10(a) is taken as an input with a higher resolution of rays in y direction as shown in (d). The generated contour from the y-dexel data is shown in (e). The generated contour from the x-dexel data is the same as (b). Since the contours in (b) and (e) have

the same topology, our contour combination algorithm can combine the two contours to reconstruct the correct shape as shown in (f).

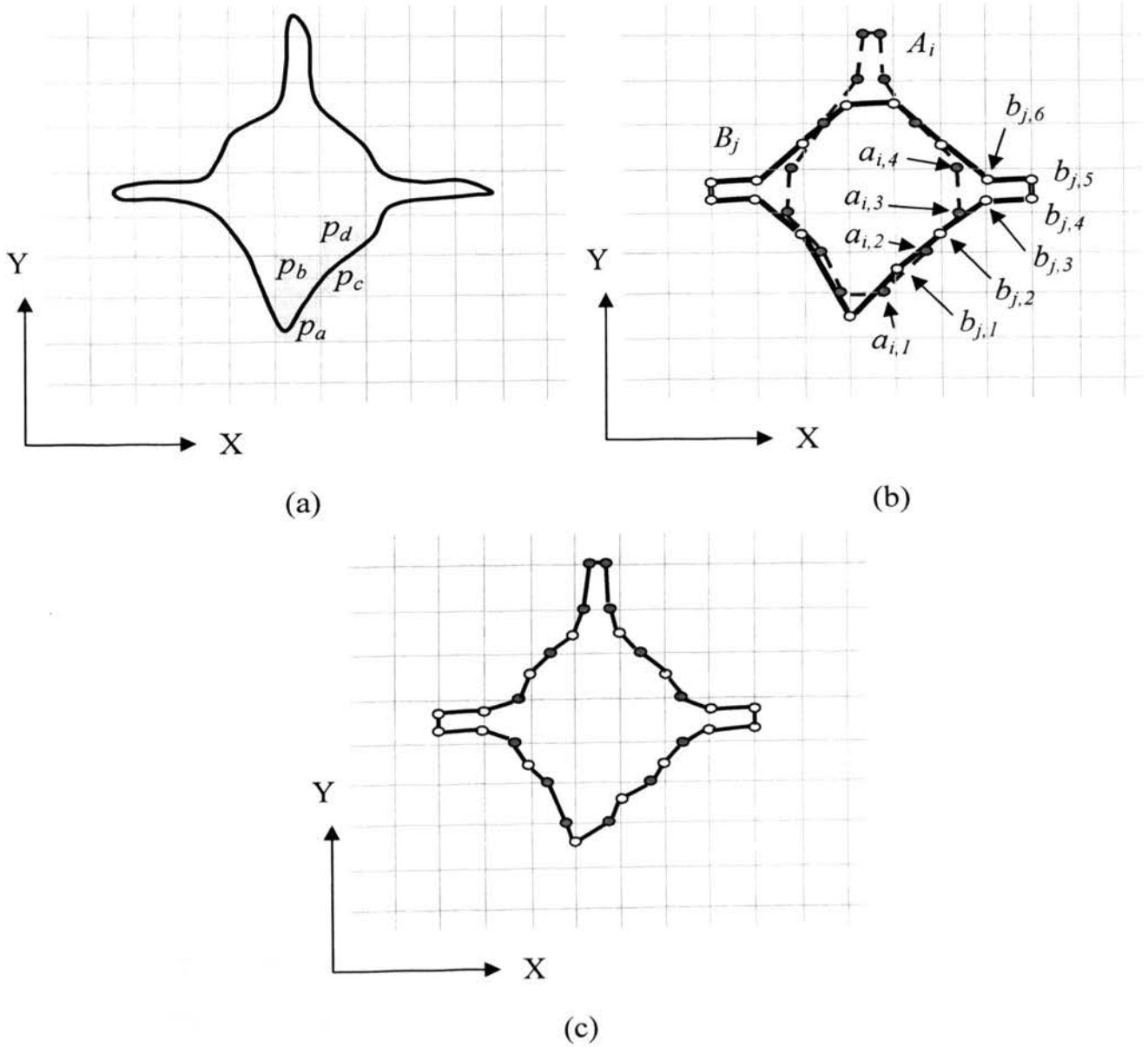


Figure 9. Example of the Contour Combination Process. (a) Original Contour, (b) Two Generated Contours from the Contour Generation Process and (c) the Combined Contour (Cont.)

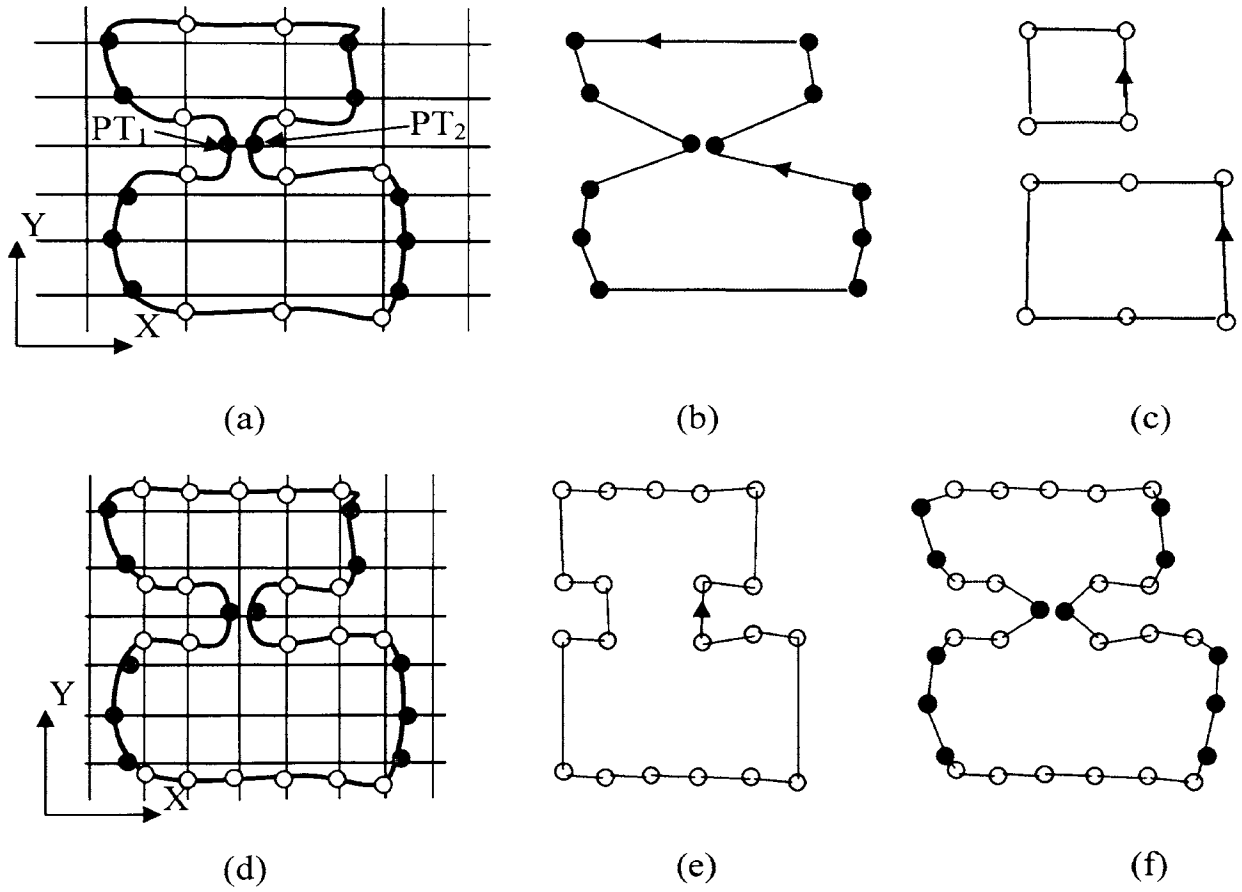


Figure 10. A Case Study of the Contour Combination Process. (a) The Input Contour with X-Dexel and Y-Dexel Data, (b) Contour Generated from the X-Dexel Data, (c) Contours Generated from the Y-Dexel Data, (d) The Input Contour with an Increase of Rays in Y Direction, (e) Contours Generated from Y-Dexel Data in (d), and (f) Combined Contour from (b) and (e) (Cont.)

Another special case is illustrated in Fig. 11. The initial shape is shown in (a). Because the space between adjacent rays is not small enough, the generated contours have neither the same topology, nor any overlapping area as shown in (b). The contour combination algorithm would again fail in this case. After increasing the number of rays in x and y directions as shown in (c), the reconstructed contours are in (d) and (e). The combined contour using our algorithm is in (f). This example again shows that the contour combination problem can be solved by increasing the density of rays.

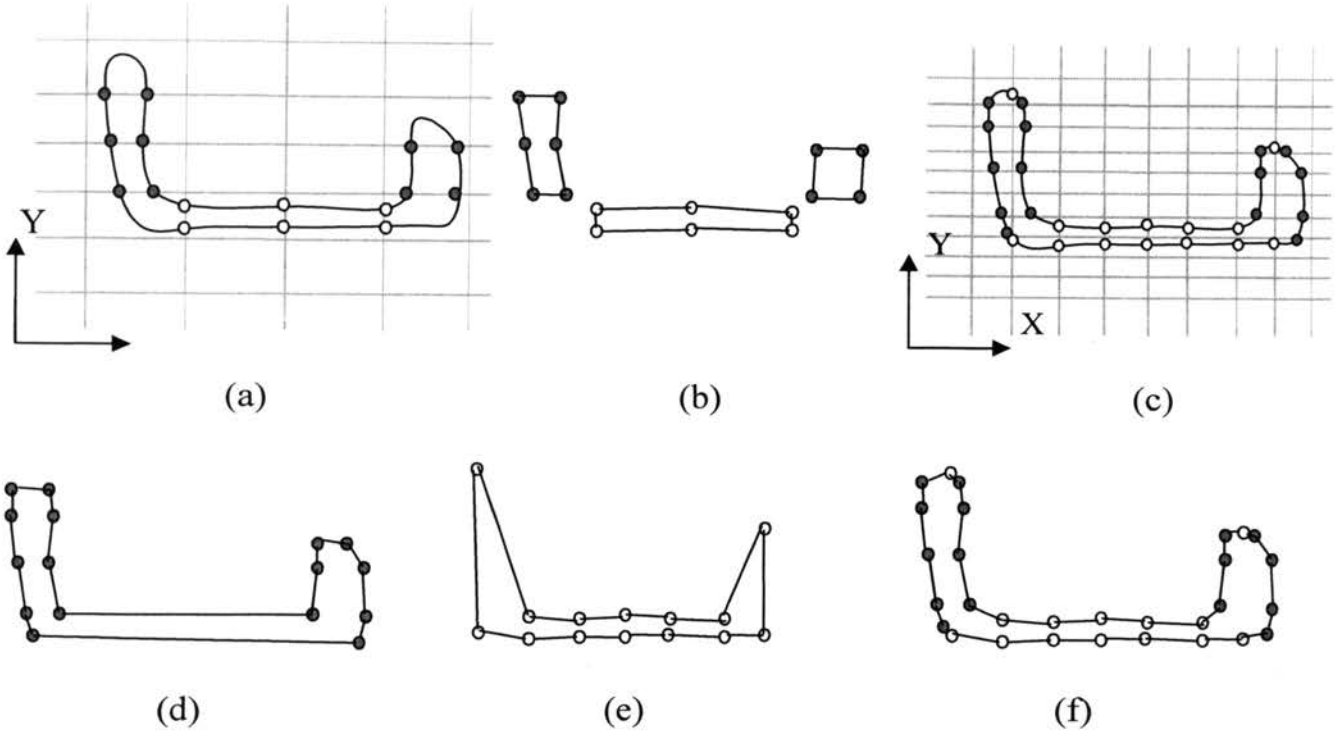


Figure 11. A Case Study of the Contour Combination Process. (a) the Input Contour with X- and Y-Dexel Data, (b) Contours Generated from the X-Dexel Data and Y-Dexel Data, (c) the Dexel Points after Increasing the Number of Rays in X and Y Directions, (d) Contours Generated from X-Dexel Data in (c), (e) Contours Generated from Y Dexel Data in (c), and (f) Combined Contour from (d) and (e)

3.3. Volume-Based Surface Tiling Algorithm

After the contour combination process, three sets of orthogonal slices of contours are generated. The volume-based tiling algorithm of Svitak and Skala [2004] is utilized to reconstruct the boundary surface of the 3D model from these contours. The main idea of this volume-based tiling algorithm is to generate triangular facets within each rectangular box associated with the rays in x, y and z directions. Because the three sets of orthogonal contours contain the positions and connectivity of all triangle vertices, the problem of generating triangular meshes within each box becomes the problem of searching the locations and connection information of the vertices from the three sets of contours that have been generated. Once this information is obtained, it is trivial to generate the triangular facets within each box by using a triangular patching algorithm.

The volume-based tiling algorithm consists of three steps. Given a triple-dexel data with M, N, and O numbers of divisions in the x, y and z axes, respectively, the 3D space is divided into $M \times N \times O$ equal-sized rectangular boxes. The algorithm first identifies the Boundary Sub-Volumes (BSVs) that are the boxes having non-null intersections between their edges and the solid's boundary surface. Second, the three orthogonal sets of contours are searched to find a close loop of vertices within each BSV. Finally, triangular facets are created within each BSV by patching these vertices.

Some details of the algorithm are given in the follows. The algorithm identifies the BSVs by searching the intersection points within the object's boundary surface along the three orthogonal sets of rays. For example, in Fig. 12(a), the intersection point between ray R1 and the object's boundary surface is point p and thus, boxes A, B, C and D are the BSVs. Within each BSV, the boundary surface of the object forms a close loop. To find the close loop of vertices within a BSV, the algorithm starts from the point on the bottom of the BSV and ends when coming back to the starting point.

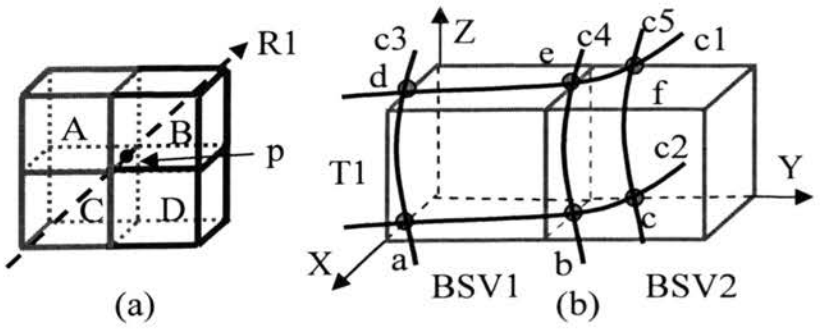


Figure 12. Volume Tiling Algorithm. (a)Identification of Boundary Sub-Volumes and (b) Generation of Surface Patches within Two Boundary Sub-Volumes

Taking Fig. 12(b) as an example, within BSV1, the search starts from point a on the bottom. After searching contour $c2$ on the xy plane, the next point found is point b. Because point b is on both contour $c2$ and contour $c4$, thus, contour $c4$ is searched to find

the next point, which is point e. The search continues to find point d and then point a, which is the sarting point. Thus, within BSV1, the close loop of points $a \rightarrow b \rightarrow e \rightarrow d \rightarrow a$ is generated and then patched into two triangular facets *abe* and *aed*. Likewise, the close loop $b \rightarrow c \rightarrow f \rightarrow e \rightarrow b$ is generated within BSV2.

4. ANALYSIS

In this section, the computation complexity and the memory requirement of the contour generation algorithm and the contour combination algorithm are analyzed. Further, the surface error is analyzed to estimate the quality of the reconstructed surface.

4.1. Computational Complexity Analysis

4.1.1. Contour Generation Algorithm

The contour generation algorithm using dexel data has four steps: (1) grouping the dexels, (2) connecting the dexel points, (3) constructing the connection table, and (4) traversing the dexel points in the table. According to the analysis described in a previous paper [Zhang et al., 2007], the computation time for Step 1 is $c_1\beta(\alpha+1)^2$, where α is the average number of dexel points along a ray, β is the number of rays intersecting with the object on a slice ($\beta >> 1$), and c_1 is a constant. In Step 2 and Step 3, the dexel points in each group are connected and the connection table is generated. The computation time of these two steps are each proportional to $\alpha\beta$. In Step 4, the points in the connection table are searched to obtain contours. The computation time of this step is also proportional to $\alpha\beta$. Thus, the total computation time of the contour generation algorithm for one slice is $c_1\beta(\alpha+1)^2 + c_2\alpha\beta$, where c_2 is a constant. In triple-dexel modeling, suppose M, N and O are the numbers of divisions in the x, y and z axes, respectively, then the x dexel data generates O number of xy slices and N number of xz slices in the contour generation. For a xy slice, since there are N number of rays in x direction, the computation time of the contour generation algorithm for the x dexel data on xy slices can be calculated by replacing β with N and taking α as the average number of dexel points per ray in the triple-dexel model, i.e., $\alpha = T/(MN + NO + OM)$, where T is the total number of dexel points of the triple-dexel model. The resultant computation time is $(c_1(\alpha+1)^2N + c_2\alpha N)O$. For the x-dexel data on xz planes, the computation time is $(c_3(\alpha+1)^2O + c_4\alpha O)N$. Thus, the complexity of the contour generation algorithm for x-dexel is $O(\alpha T_x)$ where T_x is the

number of dexel points in x-dexel data. Likewise, the computation time for y-dexel and z-dexel data can be found. Thus, the total computation complexity for the contour generation algorithm is:

$$O(\alpha T_x)+ O(\alpha T_y)+O(\alpha T_z)= O(\alpha T) \quad (5)$$

where T_y and T_z are the numbers of dexel points in x-dexel and y-dexel data, respectively.

4.1.2 Contour Correspondence Algorithm

The contour correspondence operation is done between two sets of input contours A and B on each slice. In this operation, the bounding box of each contour is calculated and the overlapping ratio between every contour from contour set A and every contour from contour set B is computed. The computation time is $c_5(P_A+P_B)+ c_6u^2$, where P_A and P_B are the numbers of points in contour sets A and B respectively, u is the number of contours in each of contour sets A and B, and c_5 and c_6 are constants. Since u^2 is much smaller in comparison with P_A and P_B, this computation time is proportional to $c_5(P_A+P_B)$.

4.1.3. Contour Combination Algorithm

The contour combination operation has two steps. The first step requires searching for the first pair of points from A_i and their associated points from B_j, and then the second step continuously searches for the rest of point pairs from A_i to identify each pair that has at least one associated point from B_j in between. Meanwhile, the associated points from contour B_j are also identified and inserted into A_i. The first step is run once for every two corresponded contours. Finding the starting pair of points in A_i that satisfies the criteria discussed in Section 3.2.3 takes $c_7P_{a,i}$, and searching for their corresponding points in contour B_j takes $c_8P_{b,j}$ time, where $P_{a,i}$ and $P_{b,j}$ are the numbers of points in contour A_i and contour B_j, respectively. The second step searches for the next pair of points in A_i which has associated points from B_j in between, identifies the associated points, and inserts these points into A_i. The time taken for this operation is $c_9P_{a,i}+ c_{10}P_{b,j}$. This process is repeated until all the points in contour B_j are added to A_i. Thus, the overall computation time for the contour combination operation for contour A_i and B_j is $(c_7+c_9)P_{a,i}+ (c_8+c_{10})P_{b,j}$. By summing the computation times for all the contours on one slice, i.e., for two sets of contours A_i (i=1,..., u) and B_j (j=1,..., u), the computation time of the contour corresponding and combination operations per slice is

$$c_5(P_A + P_B) + \sum_{i=1}^{u}(c_7 + c_9)P_{a,i} + \sum_{j=1}^{u}(c_8 + c_{10})P_{b,j} = c_{11}P_A + c_{12}P_B = c_{13}P \quad (6)$$

where P is the total number of dexel points of contour sets A and B ($P=P_A+P_B$) on a slice. Thus, the computation time of the contour correspondence and combination operations is proportional to the number of dexel points on the slice. In a triple-dexel model, the total computation time of the contour corresponding and combination operation is proportion to

$$(M)\alpha(N+O) + (N)\alpha(O+M) + (O)\alpha(M+N) = 2T = \mathrm{O}(T) \quad (7)$$

From the above analysis, it is concluded that the computation complexity of the contour generation algorithm is $\mathrm{O}(\alpha T)$ and the complexity of the contour correspondence and combination algorithms is $\mathrm{O}(T)$, where α is the average number of dexel points along a ray and T is the total number of dexel points.

4.2. Space Complexity Analysis

The contour generation algorithm stores in the computer memory (i) the initial dexel data, (ii) the connections between dexel points, and (iii) the final contour data. As mentioned before, the linked list structure is used to store the initial dexel data, which has the storage cost proportional to the number of dexel points. The connections of the dexel points and the generated contours are also each saved in a linked list structure. In the contour combination algorithm, the same linked list structure stores the combined contours on each slice. Overall, the memory costs of the contour generation and contour combination algorithms are linearly proportional to the number of dexel points of the triple-dexel model.

4.3. Surface Error Analysis

The reconstructed surface is watertight because in the volume-based surface tiling algorithm, every dexel point inside the boundary sub-volume is guaranteed to have connection points to form a close loop. Note that the connectivity information is embedded in the three orthogonal slices of contours. However, the reconstructed surface is still an approximation of the original shape. To estimate the quality of the reconstructed surface, the reconstructed surface error is defined as the ratio of the *Hausdorff* distance between the original surface and the reconstructed surface to the diagonal length of the bounding cuboid. *Hausdorff* distance is the maximum distance

between two non-empty sets of data. To calculate the surface error, the reconstructed surface model is sampled from three orthogonal directions and the *Hausdorff* distance between the sampled points and the original surface model is calculated and normalized by the diagonal length of the bounding cuboid. The normalized surface error e between the sampled points P and the original surface S is:

$$e = \frac{d_H(P,S)}{L} \tag{8}$$

where L is the diagonal length of the bounding cuboid. $d_H(P, S)$ is the *Hausdorff* distance between the set of sampled points $P = \{p_1 \ldots p_n\}$ of the reconstructed surface and the original surface model S. It is evaluated as the maximum of the distances between point set $p_i \in P$ and the surface S, i.e.,

$$d_H(P,S) = \max_{p_i \in P} d(p_i,S) \tag{9}$$

where the distance between a point in $p_i \in \mathrm{R}^3$ and a surface S is given by:

$$d(p_i,S) = \min_{q \in S} \| p_i - q \|_2 \tag{10}$$

The surface errors of the reconstructed Stanford bunny model from triple-dexel data are calculated using the Metro [Cignoni et al., 1998] comparison tool under four different resolutions as shown in Table 2. The Metro takes the original surface model and the reconstructed surface model as the input and outputs the surface error between them. It can be seen in Table 2 that the error increases to 1.332% when the resolution of the model decreases to 30×30. The surface errors of the models reconstructed from the single-dexel data and the triple-dexel data are compared in Table 3, which clearly shows that the reconstructed surface from the single-dexel data has larger errors than the surface reconstructed from the triple-dexel data for the same ray resolution.

5. IMPLEMENTATION EXAMPLES

Implementation examples of the triple-dexel based surface reconstruction process are given in this section. The plate model as shown in Fig. 13(a) is discretized into triple-dexel data. The contour generation algorithm generates two sets of contours on *xy* planes,

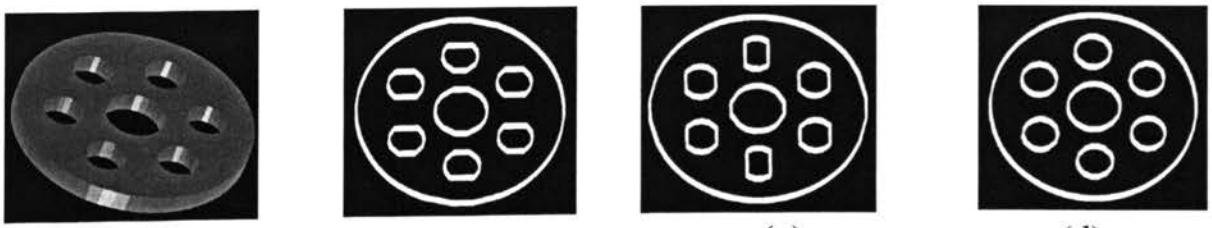
as shown in Fig. 13(b) and Fig. 13(c), from the dexel data in x and y directions, respectively. The combined contours on xy planes are shown in Fig. 13(d).

Table 2. Surface Errors of the Reconstructed Bunny Model from Triple-Dexel Data

Resolution	Hausdorff distance (d_H: mm)	Normalized error (e)
30*30	0.003334	1.332%
50*50	0.001989	0.7525%
100*100	0.001445	0.4390%
200*200	0.000789	0.2656%

Table 3. Surface Errors of the Reconstructed Bunny Model

Resolution	Normalized error from the triple-dexel model	Normalized error from the single-dexel model
50*50	0.7525%	1.365%
100*100	0.4390%	0.658%



(a) (b) (c) (d)

Figure 13. Illustrative Example of the Contour Combination Algorithm. (a) Input Object Model, (b) Contour Generated from X-Dexel Data, (c) Contour Generated From Y-Dexel Data, and (d) Combined Contour from (b) and (c)

After generating three orthogonal sets of contours in the contour combination process as described, the volume-based surface tiling algorithm is utilized to generate the boundary surface of the 3D model. Figure 14 shows the reconstructed surface of a bunny from the obtained contours on 70 slices in each of *xy*, *yz*, and *zx* planes. Figure 15 illustrates the surface improvement from the triple-dexel data over the single-dexel data. Figures 15(a) and (c) show the results of surface reconstruction from single-dexel data, and Fig. 15(b) and (d) show the corresponding results of surface reconstruction from triple-dexel data. These figures clearly show that the generated surface from the triple-dexel data is more accurate than the reconstructed surface from the single-dexel data when using the same ray resolution.

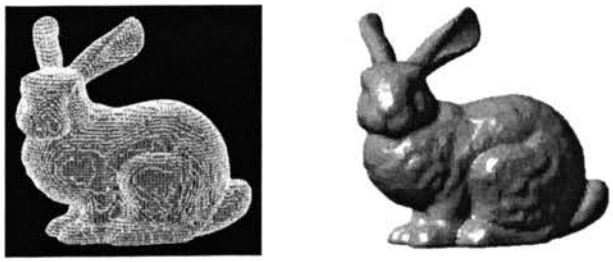


Figure 14. A Bunny Model and the Reconstructed Surface of the Bunny

The developed surface reconstruction process based on the triple-dexel model is incorporated into a virtual sculpting system [Peng and Leu, 2003; Leu et al., 2005; Peng et al., 2006]. The virtual sculpting system is developed on a Microsoft Windows XP workstation. The software is written in C++, and the graphics-rendering component is built on OpenGL and GLUT. The haptics interface is implemented using the PHANToM™ device and the GHOST (General Haptics Open Software Toolkit) SDK software available from SensAble Technologies. This virtual sculpting system enables the user to create and modify 3D freeform objects through interactive sculpting operations and gives the user real-time force feedback during the sculpting process. The

tool swept volume between two consecutive sampling times is obtained by the Sweep Differential Equation method [Blackmore and Leu, 1992] and represented by boundary triangular meshes [Peng and Leu, 2003]. The workpiece and the tool swept volumes are scan-converted to obtain their triple-dexel data. Boolean operations on the triple dexels are performed by comparing and merging the dexel data in each of x, y or z directions. The surface reconstruction software is executed during the sculpting process to convert the triple-dexel model to a triangular mesh model. Figure 16 shows the setup of the virtual sculpting system and a cat model created using the system and viewed from two different directions.

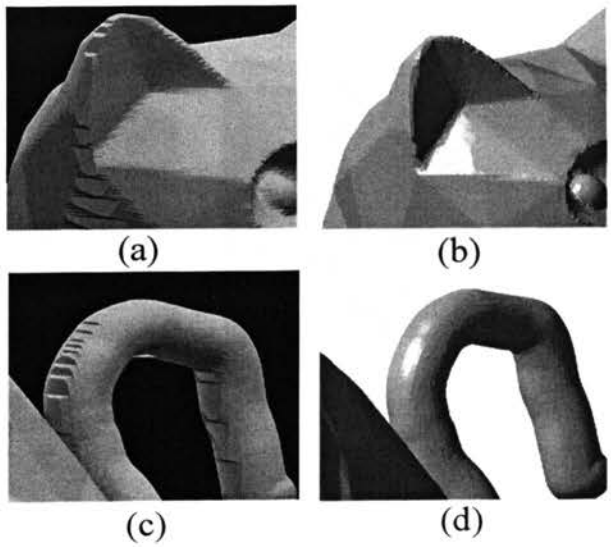


Figure 15. Comparisons Between Single-Dexel Data and Triple-Dexel Data. (a) and (c) Surfaces Reconstructed from Single-Dexel Data, (b) and (d) from Triple-Dexel Data

6. COMPARISON WITH VOXEL REPRESENTATION

Voxel modeling is a popular representation scheme [Kaufman et al., 1995; Hadwiger et al., 2006]. To benchmark the performance of the developed method, numerical experiments are conducted to compare using triple-dexel data vs. voxel data in terms of the surface reconstruction time and the associated surface error. An impeller and

a bunny model, as shown in Fig. 17, are discretized into voxel data in the resolution of 50*50*50, 100*100*100 and 150*150*150 by using a fast voxelization algorithm [Karabassi et al., 1999]. The voxel data is stored in the 3D array structure, with the marching cube algorithm utilized to reconstruct the surface from the voxel data. Meanwhile, the triple-dexel data is stored in the linked list structure and the object's surface is reconstructed using the method developed in this paper. The normalized surface errors of the reconstructed surface are calculated using the Metro comparison tool [Cignoni et al., 1998] and shown in Table 4. The time of the contour generation, correspondence and combination process is compared with the surface reconstruction time from the voxel representation in different resolutions in this table.

Figure 16. A Cat Model Generated Using the Virtual Sculpting System

The test result shows that, under the same resolution, the surface reconstructed from the triple-dexel data has a smaller surface error in comparison with the surface reconstructed from the voxel data. This is because the triple-dexel based method utilizes actual positions of the intersection points between rays and the object's boundary surface as the vertices of the reconstructed surface model, while the voxel based method approximates the positions of these vertices by voxel interpolation.

The computation complexity of the contour generation, correspondence and combination process using triple-dexel data is $O(T)$ or $O(M^2)$, where M is the number of divisions along each axis. Because the complexity of the volume-based tiling algorithm is

also $O(M^2)$[Svitak, 2004], the developed surface reconstruction method is more efficient than the voxel-based surface reconstruction method, whose computational complexity is $O(M^3)$. The total computation times for surface reconstruction from the voxel data and from the triple-dexel data of the impeller and the bunny models are plotted vs. the number of divisions along each axis in Figs. 18 and 19. The results in these figures verify that the triple-dexel model is more efficient than the voxel model.

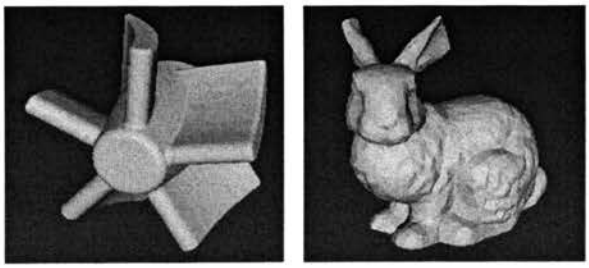


Figure 17. Two Test Cases: Impeller and Bunny

Table 4. The Surface Reconstruction Time and Surface Error

Resolution	Test Model	No. of Dexel Points	Reconstruc-tion Time Using Triple Dexels (s)	Reconstruction Time Using Voxels (s)	Error of Reconstructed Surface from Triple-Dexel Data (%)	Error of Reconstructed Surface from Voxel Data (%)
50*50*50	Impeller	19916	0.1333	0.12843	0.4263	0.9091
	Bunny	14402	0.0985	0.11755	0.7525	1.0272
100*100*100	Impeller	79632	0.4974	0.9836	0.1683	0.4012
	Bunny	57852	0.3631	0.9436	0.4390	0.5063
150*150*150	Impeller	179727	1.1720	3.2290	0.1843	0.2653
	Bunny	130650	0.8230	3.1300	0.2656	0.4299

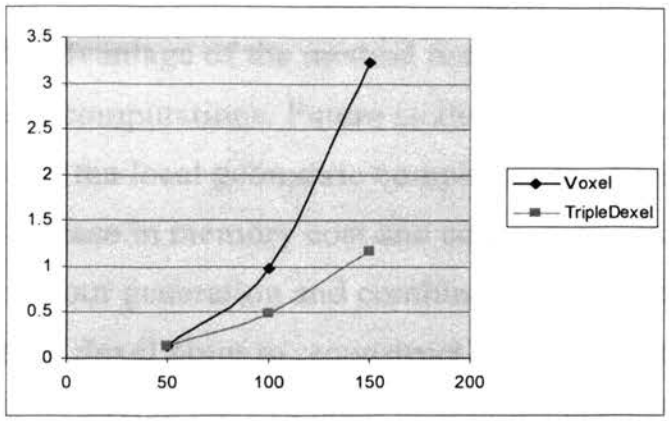


Figure 18. Surface Reconstruction Time vs. Number of Divisions from the Voxel Data and the Triple-Dexel Data for the Impeller

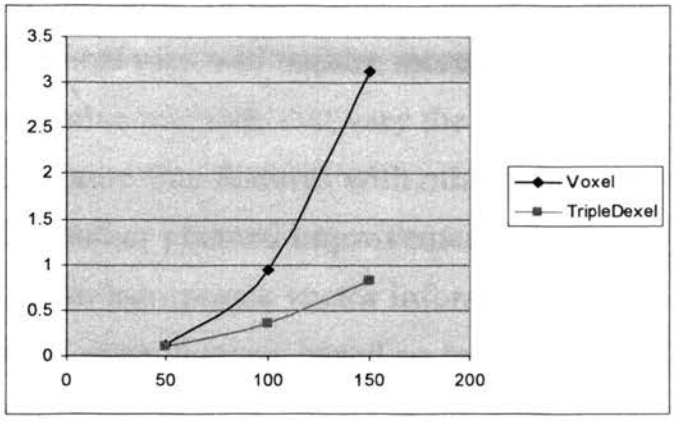


Figure 19. Surface Reconstruction Time vs. Number of Divisions from the Voxel Data and the Triple-Dexel Data for the Bunny

7. CONCLUSIONS

This paper has described a novel method of surface reconstruction from triple-dexel data. Three sets of contours on orthogonal slices are generated from triple-dexel data by a contour generation algorithm and a contour combination algorithm. A volume-based tiling algorithm is then utilized to generate the boundary surface of the 3D object in triangular patches from these contours. The computation complexity and the memory requirements of the developed method are analyzed. Both computation time and memory cost are found to be linearly proportional to the number of dexel points of the triple-dexel model. Comparing with the surfaces reconstructed from single-dexel data and from voxel data with the same resolution, our triple-dexel based method has a higher surface accuracy. Also the described surface reconstruction method is more efficient than the popular voxel-based method. The developed surface reconstruction process has been incorporated into a virtual sculpting system to address the view-dependent problem inherent in triple-dexel modeling. Examples are given to demonstrate the capability of the developed method.

The developed contour combination method requires the same numbers of input contours generated from rays in two orthogonal directions for each slice. In case the numbers of contours are different, the density of rays to scan the object needs to be

increased until the numbers of input contours are the same for each slice. This is the main disadvantage of the method because casting additional rays will require more memory and computations. Future studies will explore adaptive methods that vary the ray density with the local geometric complexity in order to capture fine features with minimal increase in memory cost and computation time. Another planned improvement of our contour generation and combination algorithms is to incorporate vector information at each dexel point to reconstruct surfaces containing sharp features based on techniques described in the extended marching cube method [Kobbelt et al., 2001] or the dual contouring method [Ju et al., 2002].

ACKNOWLEDGMENTS

This research is supported by a National Science Foundation award (CCR-0310619) and by the Intelligent Systems Center at the Missouri University of Science & Technology. The bunny model is courtesy of the Stanford 3D Scanning Repository.

REFERENCES

1. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T., 2001, "Point Set Surfaces," IEEE Visualization 2001, October, IEEE, New York, pp. 21–28.
2. Amenta, N., Choi, S., and Kolluri, R., 2001, "The Power Crust," in 6th ACM Symposium on Solid Modeling and Applications, Ann Arbor, MI.
3. Azernikov, S., Miropolsky, A., and Fischer, A., 2003, "Surface Reconstruction of Freeform Objects Based on Multi-resolution Volumetric Method," Proc. of ACM Symposium on Solid and Physical Modeling, pp. 115-126.
4. Barequet, G., and Vaxman, A., 2007. "Nonlinear Interpolation between Slices," Proc. of the ACM Symposium on Solid and Physical Modeling, Beijing, China, June 04-06, pp. 97-107.
5. Benouamer, M.O., and Michelucci, D., 1997, "Bridging the Gap between CSG and Brep via a Triple Ray Representation," Proc. of Solid Modeling'97, Atlanta, GA, May 14-16, pp. 68-79.

6. Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G., 1999, "The Ball-Pivoting Algorithm for Surface Reconstruction," IEEE Trans. Vis. Comput. Graph., 5(4), pp. 349–359.
7. Blackmore, D., and Leu, M. C., 1992, "Analysis of Swept Volume via Lie Groups and Differential Equations," International Journal of Robotics Research, 11(6), pp. 516-537.
8. Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R., 2001, "Reconstruction and Representation Of 3D Objects With Radial Basis Functions," Proc. of the 28th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '01. ACM, New York, NY, 67-76.
9. Cignoni, P., Montani, C., and Scopigno, R., 1998, "A Comparison of Mesh Simplification Algorithms," Computer Graphics, 22(1), pp. 37–54.
10. Dey, T., Giesen, J., and Hudson, J., 2001, "A Delaunay Based Shape Reconstruction From Large Data," Proc. of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics, pp. 19-27.
11. Edelsbrunner, H., and Mucke, E. P., 1994, "Three-dimensional Alpha Shapes," ACM Transactions on Graphics, 13(1), pp. 43–72.
12. Hadwiger, M., Kniss, J. M., Rezk-salama, C., and Weiskopf, D., 2006, Real-time Volume Graphics, A. K. Peters.
13. Huang, Y., and Oliver, J. H., 1995, "Integrated Simulation, Error Assessment and Tool Path Correction for Five-Axis NC Milling," Journal of Manufacturing Systems, 14(5), pp. 331-344.
14. Ju, T., Losasso, F., Schaefer, S., and Warren, J., 2002, "Dual Contouring of Hermite Data," Proc. of SIGGRAPH'2002, pp. 339-346.
15. Leu, M. C., Peng, X., and Zhang, W., 2005, "Surface Reconstruction for Interactive Modeling of Freeform Solids by Virtual Sculpting," Annals of the CIRP, 54(1), pp. 131-134.
16. Lorensen, W.E., and Cline, H.E., 1987, "Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm," Computer Graphics, 21(4), pp. 163-169.
17. Karabassi, E. A., Papaioannou, G., and Theoharis, T., 1999, "A Fast Depth-Buffer-Based Voxelization Algorithm", Journal of Graphics Tools, 4(4), pp. 5-10.

18. Kaufman, A., Cohen, D., and Yagel, R., 1995, "Volume Graphics," IEEE Computer, 26(7), pp. 51-64.
19. Kobbelt, L. P., Botsch, M., Schwanecke, U., and Seidel, H. 2001, "Feature Sensitive Surface Extraction From Volume Data," Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '01. ACM, New York, NY, pp. 57-66.
20. Konig, A. H., and Groller, E., 1998, "Real-Time Simulation and Visualization of NC Milling Processes for Inhomogeneous Materials on Low-End Graphics Hardware," Proc. of the Computer Graphics International, IEEE Computer Society, pp. 338.
21. Muller, H., Surmann, T., Stautner, M., Albersmann, F., and Weinert, K., 2003, "Online Sculpting and Visualization of Multi-Dexel Volumes," Proc. of ACM Symposium on Solid and Physical Modeling, Seattle, Washington, June 16-20, pp. 258-261.
22. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H. 2003, "Multi-level Partition of Unity Implicits," ACM Trans. Graph. 22(3), pp. 463-470.
23. Ohtake, Y., Belyaev, A., and Seidel, H. 2006, "Sparse Surface Reconstruction with Adaptive Partition of Unity and Radial Basis Functions," Graph. Models, 68(1), pp. 15-24.
24. Peng, X., and Leu, M.C., 2003, "Interactive Solid Modeling in a Virtual Environment with Haptic Interface," Virtual and Augmented Reality Applications in Manufacturing, Springer-Verlag London Limited, London, UK.
25. Peng, X., Zhang, W., and Leu, M.C., 2006, "Freeform Modeling Using Sweep Differential Equation with Haptic Interface," Journal of Virtual and Physical Prototyping, 1(3), pp. 183-196.
26. Pfister H., Zwicker, M., Van Baar, J., and Gross, M., 2000, "Surfels: Surface Elements as Rendering Primitives," Proc. of SIGGRAPH 2000, pp. 335-342.
27. Ren, Y., Lai-Yuen, K.S., and Lee, Y.S., 2006, "Virtual Prototyping and Manufacturing Planning by Using Tri-dexel Models and Haptic Force Feedback," Virtual and Physical Prototyping, 1(1), pp. 3-18.
28. Svitak, R., and Skala, V., 2004, "A Robust Technique for Surface Reconstruction from Orthogonal Slices," Machine Graphics & Vision, 12(3), pp. 221-233.

29. Van Hook, T., 1986, "Real-Time Shaded NC Milling Display," Proc. of ACM SIGGRAPH, ACM Press, pp. 15-20.
30. Wang, Y. F., and Aggarwal, J. K., 1986, "Surface Reconstruction and Representation of 3D Scenes," Pattern Recognition, 19(3), pp. 197-207.
31. Zhang, W., Peng, X., Leu, M.C., and Zhang, W., 2007, "A Novel Contour Generation Algorithm for Surface Reconstruction from Dexel Data," Journal of Computing and Information Science in Engineering, 7(3), pp. 203-210.

III: VIRTUAL SCULPTING WITH SURFACE SMOOTHING BASED ON LEVEL-SET METHOD

Weihan Zhang and Ming C. Leu

Department of Mechanical and Aerospace Engineering
Missouri University of Science & Technology
Rolla, Missouri 65409, USA
Email: wzxq6@mst.edu, mleu@mst.edu

ABSTRACT

This paper presents a surface smoothing technique based on the level-set method. The triple-dexel data used to represent the generated model in virtual sculpting is converted into distance field data by identifying spatial grid points close to the model's boundary surface and calculating their Euclidean distance values. The surface is smoothed by solving the level-set differential equation with mean curvature flow using a fast and robust numerical scheme. Examples are given to demonstrate the effectiveness of the surface smoothing operation for virtual sculpting.

Keywords: Computer Aided Design, Surface Smoothing, Level-Set Method

1. INTRODUCTION

More and more products with complex geometries are being designed and manufactured by computer aided design (CAD) and rapid prototyping (RP) technologies. Freeform surface is one of the geometrical features widely used in modern products like car bodies, airfoils and turbine blades as well as in aesthetic artifacts. How to efficiently design and generate digital prototypes with freeform surfaces is an important issue in CAD. None-Uniform Rational B-Splines (NURBS) is an industrial standard for freeform surface design. However, generating a NURBS surface of complex geometry requires creating and positioning a large number of control points using 2D input devices like

mouse and keyboard, which is a tedious work and is not highly intuitive. Virtual sculpting is a process in which the user creates a three-dimensional (3D) object on the computer screen by interactively carving a virtual workpiece like a real sculptor would do on a piece of clay, wax or wood. It is well suited to freeform design of virtual prototypes as it allows the user to avoid cumbersome mouse and keyboard interface.

Various techniques [1, 2] have been developed for freeform shape design with a haptic device. Our past research has contributed to this topic by developing a dexel based virtual sculpting system capable of removing and adding materials for the creation of freeform shapes in real time [3, 4]. The "view-dependence problem" of dexel representation has been recently solved by developing contour generation, contour combination, and surface tiling algorithms to reconstruct the boundary surface of the sculpted solid from triple-dexel data [5]. However, that virtual sculpting system did not have surface smoothing capability. It is very desirable to enable the user to smooth the rough area created during the sculpting process. The level-set method with mean curvature flow is a surface smoothing technique that has been researched. It always produces none self-intersecting surfaces that represent physically realizable objects. However, previous studies on the use of level-set methods for surface smoothing can only calculate the underlying distance field data from a triangular mesh, not from triple-dexel data.

In the present paper, we build upon our recent work of surface reconstruction from triple dexels [5] to develop a method to calculate the distance field directly from triple-dexel data for surface smoothing in virtual sculpting. With the 3D distance field data, the virtual sculpting system is further developed to include surface smoothing operations based on the level-set method. The level-set differential equation with mean curvature flow is solved using a fast and robust numerical scheme to smooth the boundary surface for any user selected area. The developed method seamlessly integrates level-set based surface smoothing into the virtual sculpting system.

This paper is organized as follows. Section 2 provides an overview of related research work. In Section 3, we present the method of distance field calculation from triple-dexel data. Section 4 describes the surface smoothing operation based on the level-

set method. Modeling examples in virtual sculpting are shown in Sections 5. Conclusions are drawn in Section 6.

2. RELATED WORK

2.1. Distance Field Calculation

The distance field is defined as a spatial function which returns the signed Euclidean distance from a spatial point (x,y,z) to the boundary, ∂M, of a manifold object M. The sign denotes whether the point is inside or outside ∂M. The calculation of distance field from a triangular mesh has been extensively studied. A survey of research on this topic is available from [6]. Generally a brute force method is used to compute the distances from a grid point in the space to every boundary triangle of M and select the shortest one. To reduce the computation, the shortest distance can be calculated only to a limited number of primitives according to spatial coherences. However, there has been very little research on the calculation of the distance field directly from triple-dexel data, which precedes the creation of a triangular mesh in virtual sculpting. Sealy and Novins [7] approximated the Euclidean distance of a grid point as the shortest distance among its three axial distances. But this approximation is not accurate especially where sharp features are present.

2.2. Surface Smoothing

The objective of surface smoothing is to modify a surface to make it more functional or aesthetically pleasing. Smoothing techniques have been proposed in the context of surface fairing, where a fairness or penalty function that favors a smooth surface is minimized. The level-set method [8] with mean curvature flow provides a numerical mechanism for surface smoothing, which modifies the surface area represented as time varying iso-values of a function by solving a partial differential equation on the 3D grid. This method has several benefits in surface smoothing including the following: no self-intersection, thus, guaranteeing the generation of a simple, reliable close surface; easy change of topology for freeform shape design; free of edge connectivity and mesh quality problems associated with mesh models. A set of surface editing operators like smoothing, blending, sharpening, opening/closings, and embossing has been developed

using the level-set method [9]. We will apply the level-set method with mean curvature flow to develop surface smoothing operation in Section 4.

3. GENERATING DISTANCE FIELD FROM TRIPLE-DEXEL DATA

In our triple-dexel based virtual sculpting system, the representation of a solid during the sculpting process is by computing intersections between the solid and rays in three orthogonal (e.g., x, y and z) directions. For each ray, the intersection points and the surface normal sampled at each point are stored. Two intersection points in a line segment that is completely inside the solid is defined as a dexel. An illustration is given in Figure 1.

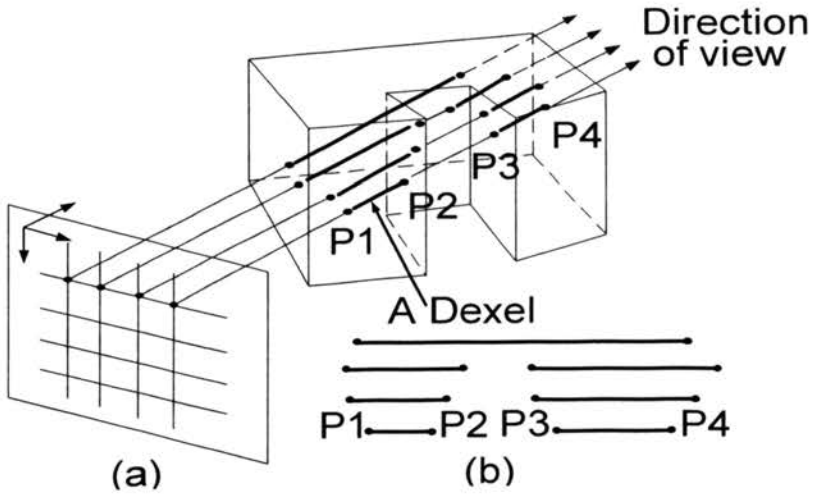


Figure 1. Illustration of the Ray-Casting Process and the Dexel Representation

To simulate the material removal process, Boolean operations are performed between the triple-dexel data of the workpiece and the tool. To visualize the sculpted solid, we have developed a surface reconstruction method from triple-dexel data. The method includes contour generation, contour combination and surface tiling algorithms [5]. Meanwhile, the PHANToM^{TM} manipulator (SensAble Technologies) is used to

provide the position and orientation data of the sculpting tool as well as haptic sensation to the user's hand during the sculpting process.

To utilize the level-set method with mean curvature flow for surface smoothing, the triple-dexel data is converted into distance field data where each grid point contains a distance value to the iso-surface representing the boundary. Formally, the signed Euclidean distance function of a grid point $(x,y,z) \in R^3$ is defined as:

$$f(x,y,z) = \mathrm{dis}[(x,y,z),S] \tag{1}$$

where S is the iso-surface and 'dis' is the Euclidean distance to S. A positive sign represents the point outside S and a negative sign represents the point inside S. The initialization of the level-set method requires the distance values of a narrow-band of grid points that are in the neighborhood of S.

We develop a four-step process for generating the distance field. First the voxels that have non-null intersections with the solid's boundary surface are identified as the Boundary Voxels (BVs). The grid point on any edge of a BV is a Boundary Grid Point (BGP) and a grid point is an Adjacent Grid Point (AGP) if it is adjacent to any BGP. Next, the sign of the distance value of each BGP and AGP is determined. Third, the surface within each BV is approximated using triangular facets. Finally, the distance value of each BGP and AGP is calculated. The details of the algorithm are given below.

3.1. Identify BV, BGP and AGP

The 3D space is divided by rays cast in three orthogonal directions into many equal sized voxels. According to the definition above, a BV must contain at least one dexel point on its edges. By scanning the dexels points along rays in the x, y, and z directions, we can find BVs. A 2D illustration is given in Figure 2, where the gray-colored pixels surrounding the iso-surface are the boundary pixels (i.e., 2D BVs). Each squared point is a BGP and each triangular point is an AGP.

3.2. Determine the Sign of Each BGP and AGP

If a grid point is between two adjacent dexels along a ray, the distance value of this point is positive. Otherwise, the sign of the distance value is negative. Thus, we can determine the sign of the distance value of each BGP and AGP by its relative position to dexels along any directional rays. It is noted that if the distance of the grid point equals zero, this grid point is on the surface. We use 0 to label this type of grid points.

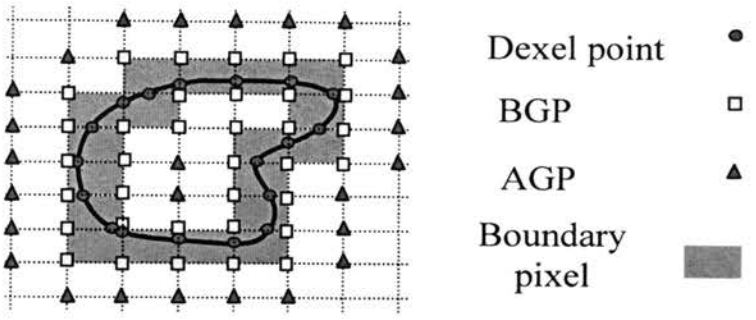


Figure 2. Boundary Pixels, BGP, AGP and Dexel Points

3.3. Approximate Iso-Surfaces Inside BVs

The main idea of this step is to use the Hermite data (i.e., exact intersection points and normals) on the edges of a BV to calculate an additional point inside the BV by minimizing a quadratic function. By connecting this point with other additional points in adjacent BVs, triangular meshes can be generated with a simple patching algorithm to approximate the boundary surface.

In the case of using triple-dexel data, the dexel points and the surface normals at these points are available from the triple-dexel data. The additional point inside a BV is the intersection point between the tangent elements of the dexel points on the edges of a BV. A 2D example is shown in Figure 3, where the circle points are the dexel points and the square points are the additional points.

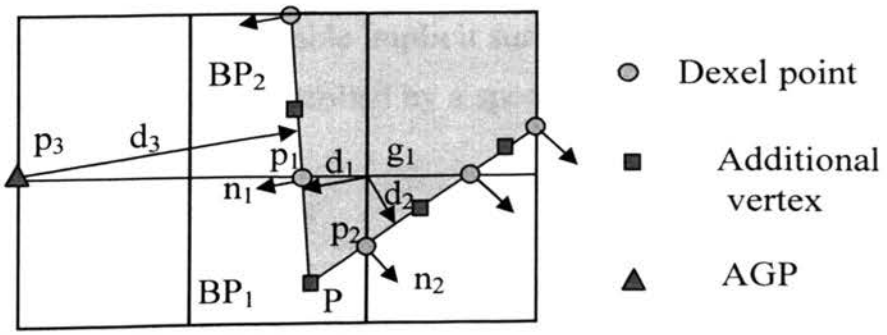


Figure 3. Distance Calculation for the Grid Points

Inside BP_1, the additional point P is the intersection point between the tangent line of dexel point p_1 under normal n_1 and the line of dexel point p_2 under normal n_2. In the 3D case, the additional point is the intersection point of three tangent planes of these dexel points on three edges of a BV. In general, this method is over-determined since more than three dexel points may exist on the edges of a BV in a 3D case. In such case, a quadratic function is minimized to find the addition point [10], i.e.

$$E(x) = \min \sum_i (n_i \cdot (x - p_i))^2 \tag{2}$$

where p_i and n_i represent dexel points and their associated unit surface normals. There exist numerical schemes to solve the least square optimization problem [11]. Once the additional vertex is generated within every BV, for each edge that contains a dexel point, the additional points of the four BVs containing the edge can be connected and patched into triangles.

3.4. Calculate Distance Values of BGPs and AGPs

The Euclidean distance of a BGP of a BV is the shortest distance from the BGP to the local triangles formed by the additional point of this BV. We calculate the distance between this BGP and every such triangle, and the smallest value is the Euclidean distance. As illustrated in Figure 3, the distance of the center grid point is d_2 because $d_2 < d_1$. Based on the same principle, to calculate the distance values of AGPs, such as point p_3 in Figure 3, only triangles formed by the additional points in the adjacent BVs are considered for the distance test.

4. SURFACE SMOOTHING USING THE LEVEL-SET METHOD

4.1. Level-Set Method

Level-set models are deformable implicit surfaces where the deformation of the surface in its normal direction is controlled by a speed function in the level-set partial differential equation [9], i.e.

$$\frac{\partial F}{\partial t} = -\nabla F \cdot \bar{v} \tag{3}$$

where $F(\mathbf{x},t)$ is the Euclidean distance function, $\mathbf{x}$ is the grid coordinates in space R^3, $\bar{v}$ is the velocity function of boundary points, ∇ is the gradient and

$$\nabla = i \cdot \frac{\partial}{\partial x} + j \cdot \frac{\partial}{\partial y} + k \cdot \frac{\partial}{\partial z} \tag{4}$$

where i, j and k are the unit vectors in R^3.

4.2. Numerical Solutions for Level-Set Method

An up-wind computation scheme [9] can be applied to solve the level-set equation. The first-order space approximation of Equation (3) is written as:

$$F_{i,j,k}^{n+1} = F_{i,j,k}^{n} - \Delta t[\max(v_{i,j,k}^{n},0)\nabla^{+} + \min(v_{i,j,k}^{n},0)\nabla^{-}] \tag{5}$$

where $v_{i,j,k}$ is the speed at a point indexed by i, j and k and

$$\nabla^{+} = \begin{bmatrix} \max(D_{i,j,k}^{-x},0)^2 + \min(D_{i,j,k}^{+x},0)^2 + \\ \max(D_{i,j,k}^{-y},0)^2 + \min(D_{i,j,k}^{+y},0)^2 + \\ \max(D_{i,j,k}^{-z},0)^2 + \min(D_{i,j,k}^{+z},0)^2 \end{bmatrix}^{1/2} \tag{6}$$

$$\nabla^{-} = \begin{bmatrix} \max(D_{i,j,k}^{+x},0)^2 + \min(D_{i,j,k}^{-x},0)^2 + \\ \max(D_{i,j,k}^{+y},0)^2 + \min(D_{i,j,k}^{-y},0)^2 + \\ \max(D_{i,j,k}^{+z},0)^2 + \min(D_{i,j,k}^{-z},0)^2 \end{bmatrix}^{1/2} \tag{7}$$

where $D_{i,j,k}^{+x}$ is a shorthand notation of the forward difference operator $\frac{F_{i,j,k}(x+h,t) - F_{i,j,k}(x,t)}{h}$ and $D_{i,j,k}^{-x}$ is the backward difference operator $\frac{F_{i,j,k}(x,t) - F_{i,j,k}(x-h,t)}{h}$.

The calculation of the level-set method can be sped up with a narrow-band scheme [9]. The idea of this method is to update only a narrow-band of grid points which are close to the iso-surface, e.g. the BGP points and AGP points in Figure 2. Another advantage of this approach is that the number of points being computed is small so that it is feasible to use a linked-list structure to keep track of them for real-time applications. For example, the BGP points can be saved into a list, and the AGP points inside the boundary and outside the boundary can be saved into two different lists. By updating the distance values of the grid points in these three lists, the change of the iso-surface is tracked.

4.3. Mean Curvature Calculation

The mean curvature (H) at a point $p \in S$ is the average of the two principal curvatures (κ_1 and κ_2)

$$H = (\kappa_1 + \kappa_2)/2 \tag{8}$$

For a 3D surface defined as a function of three coordinates, $F(x,y,z)$, the mean curvature at a grid point is

$$H = \frac{(F_{yy}+F_{zz})F_x^2 + (F_{xx}+F_{zz})F_y^2 + (F_{xx}+F_{yy})F_z^2 - 2(F_xF_yF_{xy} + F_xF_zF_{xz} + F_yF_zF_{yz})}{2(F_x^2+F_y^2+F_z^2)^{3/2}} \tag{9}$$

where the differential terms are approximated using the first-order, central finite difference numerical scheme, i.e,

$$F_x = \frac{F_{i+1,j,k} - F_{i-1,j,k}}{2\Delta x} \tag{10}$$

$$F_{xx} = \frac{F_{i+1,j,k} - 2F_{i,j,k} + F_{i-1,j,k}}{\Delta x^2} \tag{11}$$

$$F_{xy} = \frac{F_{i+1,j+1,k} - F_{i+1,j-1,k}}{4\Delta x\Delta y} + \frac{F_{i-1,j-1,k} - F_{i-1,j+1,k}}{4\Delta x\Delta y} \tag{12}$$

4.4. Surface Smoothing Using Mean Curvature Flow

If the speed (v) of a boundary point in Equation (3) is proportional to the mean curvature of the local boundary calculated by Equations (9)-(12), then Equation (3) can be written as

$$\frac{\partial F(\mathrm{x},t)}{\partial t} - b(\mathrm{x},t)H(\mathrm{x},t)\,\|\nabla F(\mathrm{x},t)\| = 0 \tag{13}$$

where $b(\mathrm{x},t)$ is a user defined function to control the speed. According to this equation, the part of the boundary with larger curvature moves faster than the part of the boundary with smaller curvature in the surface normal direction. This movement results a smoothing operation as illustrated in Figure 4, where (b)-(c) show the global smoothing of a 2D star shape in (a), and (d) shows the smoothing of the star shape locally by defining an effective area through $b(\mathbf{x},t)$.

5. APPLICATION TO VIRTUAL SCULPTING

Our virtual sculpting system runs on a Microsoft Windows XP workstation equipped with a 1.6 GHz CPU and 1 GB RAM. The software is written in C++, and the graphics-rendering component is built upon OpenGL and GLUT libraries. The haptics interface is implemented using the PHANToM™ device and the GHOST (General Haptics Open Software Toolkit) SDK software available from SensAble Technologies. Figure 5 shows the setup of the virtual sculpting system and a cat model created with the system. In the sculpting process, both the tool and the stock (initial workpiece) are represented by polyhedral boundaries. The tool location is specified by a translation and a rotation tracked by the PHANToM, and the tool swept volume between two consecutive sampling times is calculated and represented by triangular meshes. The virtual sculpting process continuously performs Boolean operations between the tool swept volume and the workpiece and computes the triple-dexel data that represents the workpiece being sculpted.

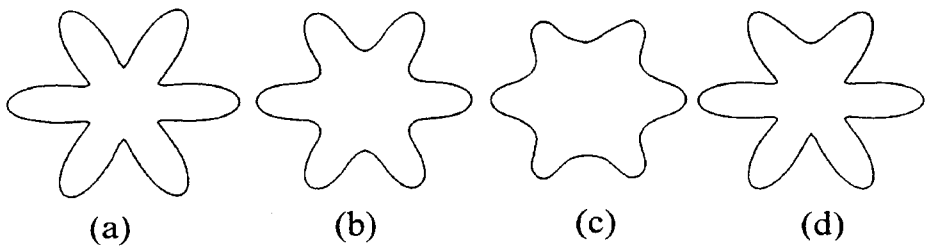


Figure 4. Example of the Smoothing Operation on a Star Shape. (a), (b)-(c) Illustrate Global Smoothing Operation and (d) Illustrates Local Smoothing Operation

Next we demonstrate the developed surface smoothing operation. In this operation, a user-defined box or sphere can be used to select a certain area on the sculpted model. Then the selected area is smoothed according to its curvature values. By adjusting $b(\mathbf{x},t)$ in Equation (13) using a scrollbar, the user is able to adjust the speed of curvature flow. Also the surface propagation process can be stopped at any time once a

satisfied result has been obtained. Figures 6(a) and (d) show two spheres joined together before and after the smoothing operation. A snowman model is created by adding and removing materials w.r.t. the two spheres model. Figures 6(b) and (c) show the snowman model and the smoothed model is shown in (e) and (f) for comparison.

Figure 5. A Cat Model Created Using the Virtual Sculpting System

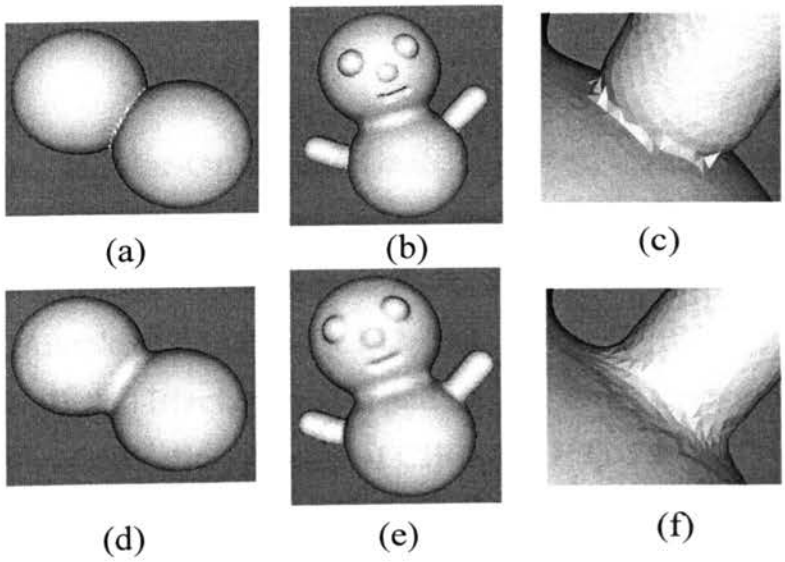


Figure 6. Modeling Examples. (a) and (d) Two Joined Spheres, (b) and (e) a Snowman Model,(c) and (f) the Boundary with and without Smoothing

6. SUMMARY

This paper has presented a level-set based surface smoothing method with application to virtual sculpting. The triple-dexel data representing a solid model is converted to distance field data by approximating the iso-surface inside the boundary voxels and calculating the Euclidean distance values for a narrow-band of grid points. The mean curvatures of the grid points in the narrow-band are estimated using the first-order finite difference numerical scheme. By using the up-wind computation scheme to solve the level-set differential equation with mean curvature flow, the higher curvature area of the boundary surface propagates faster in the surface normal direction, resulting a smoothing operation. Examples are given to demonstrate the effectiveness of the developed surface smoothing technique for virtual sculpting.

ACKNOWLEDGMENTS

This research is supported by a USA National Science Foundation award (CCR-0310619) and by the Intelligent Systems Center at Missouri University of Science and Technology.

REFERENCES

[1] Lu, S. C., Shpitalni, M., Bar-Or, R., and Gadh, R., 1999, Virtual and Augmented Reality Technologies for Product Realization, Annals of the CIRP, 48/2: 471-495.

[2] Krause, F. L., and Biahmou-Tchebetchou, A. R., 2005, Advanced Methods for a Realistic Styling, Annals of CIRP, 54/1:143-146.

[3] Leu, M.C., Maiteh, B. Y., Blackmore, D., and Fu, L., 2001, Creation of Freeform Solid Models in Virtual Reality, Annals of CIRP, 50/1:73-76.

[4] Leu, M.C., Peng, X., and Zhang, W., 2005, Surface Reconstruction for Interactive Modeling of Freeform Solids by Virtual Sculpting, Annals of CIRP, 54/1: 131-134.

[5] Zhang W., and Leu, M. C., 2007, Surface Reconstruction from Triple Dexel Model for Virtual Sculpting, Proc. of ASME International Design Engineering Technical Conference & Computers and Information in Engineering Conference.

[6] Jones, M. W., Baerentzen, J. A., and Sramek, M., 2006, 3D Distance Fields: A Survey of Techniques and Applications, IEEE Transactions on Visualization and Computer Graphics, 12/4:581-599.

[7] Sealy, G. and Novins, K., 1999, Effective Volume Sampling of Solid Models using Distance Measures, Proc. of the international Conference on Computer Graphics.

[8] Museth, K., Breen, D. E., Whitaker, T., and Barr, A. H., 2002, Level Set Surface Editing Operators, ACM Transactions on Graphics (TOG), 21/3: 330-338.

[9] Sethian, J. A., 1999, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science (2nd Ed.), Cambridge University Press, UK.

[10] Ju, T., Losasso, F., Schaefer, S., and Warren, J., 2002, Dual Contouring of Hermite Data, Proc. of SIGGRAPH.

[11] Golub, G., Loan, C. V., 1996, Matrix Computations, 3rd, Johns Hopkins Univ. Press, USA.

IV: A SPATIAL WARPING METHOD FOR FREEFORM MODELING BASED ON LEVEL-SET METHOD

Weihan Zhang and Ming C. Leu
Department of Mechanical and Aerospace Engineering
Missouri University of Science and Technology
Rolla, Missouri 65409, USA
Email: wzxq6@mst.edu

ABSTRACT

Providing an intuitive and effective tool for freeform geometric modeling is important for product design. We introduce in this paper a level-set based spatial warping method for freeform modeling, allowing shape deformation to be initialed by rigid body transformations of volumetric tools. Intuitive user operations including the imprinting, deformation and smoothing operations are developed to shield the user from the underlying geometric complexity. Unlike mesh-based spatial warping methods, the developed method represents a digital model by implicit distance field data and describes its change of geometry by the level-set method. This guarantees the generation of topologically correct triangular mesh models and circumvents the error-prone remeshing and mesh-repairing processes, thus preventing topological errors such as self-intersections. We present this method with algorithm details, numerical experiments and modeling examples.

1. INTRODUCTION

More and more products with complex geometries are being developed by computer aided design (CAD) and rapid prototyping (RP) technologies. Freeform surface is a geometrical feature widely used in modern products like car bodies, airfoils and turbine blades as well as in sculptures and other aesthetic artifacts. How to efficiently design and generate digital prototypes with freeform surfaces is an important issue in Computer Aided Geometric Design (CAGD).

None-Uniform Rational B-Splines (NURBS) is an industrial standard for freeform surface design. However, generating a NURBS surface of complex geometry requires creating and positioning a large number of control points using 2D input devices like the mouse and the keyboard. This is a tedious task and is not highly intuitive. A more effective method for freeform geometric modeling is via the Free-From Deformation (FFD) technique [Sederberg and Parry, 1986], which involves warping a space that contains the object to be modeled. However, the control lattice used for space manipulation in the FFD technique is not directly related to the object. Another important type of freeform modeling techniques is based on implicit geometric representations [Bajaj et al., 1997]. However, providing local geometric modification capability is difficult in these techniques because modifying implicit functions is not very intuitive. Recently, spatial warping methods [Gain and Marais, 2005; Angelidis et al., 2006] have become popular due to their intuitive user interface and effective modeling capabilities. All of the previous studies on these modeling techniques rely on a mesh model as the underlying geometric representation. Although mesh modeling is supported by computer hardware for fast processing, it has the problem of generating self-intersected models, which requires special care in the transformation process.

In this paper, we develop a spatial warping method based on the implicit shape representation and the level-set method to describe the geometric change of a 3D shape. We develop a grab-and-drag technique to allow the user to select different virtual tools and modify the 3D shape using selected tools through a force reflecting device. Various freeform modeling operations such as imprinting, deformation and smoothing have been developed by using rigid transformations of virtual tools. Compared with previous spatial warping techniques, the topology of the generated model from our technique is inherently correct without any self-intersection problems. Furthermore, the modeling operations developed using this technique are effective for designing freeform models and intuitive for common users.

The rest of the paper is organized as follows. Section 2 provides a review of related research work on freeform geometric modeling. The spatial warping method is introduced in Section 3. In Section 4, we describe the development of freeform modeling

operations using the spatial warping method. Modeling examples based on the spatial warping method are shown in Section 5. Conclusions are given in Section 6.

2. RELATED WORK

2.1. Freeform Geometric Modeling

Lattice based freeform deformation is an important class of deformation methods. Sederberg and Parry [1986] introduced the concept of freeform deformation. This technique defines freeform deformation by specifying a trivariate Bézier solid. The control points of the lattice can be displaced. Several improvements and extensions have been made since then. The extended freeform deformation method proposed by Coquillart [1990] utilized non-parallelepipedical lattices. Hsu et al. [1992] developed a direct manipulation technique that makes generation and placement of deformations easier. Lamousin and Waggenspack [1994] described a system of NURBS-based freeform deformation based on a mesh built from rectangular parallelopipeds. Polygon mesh based freeform deformation is another class of deformation methods. Parent [1977] initialed the use of basic vertex movement and decay function techniques. Leblanc et al. [1991] did some improvements on the decay function, interface and polygonal modeling. Bill et al. [1995] presented a polygonal modeling system using virtual sculpting tools and mesh refinement operations. The user controls a virtual tool to sculpt a freeform polygonal model starting from a mesh structure. The system allows the user to push, pull and deform the mesh in a variety of ways. A main problem of lattice based freeform modeling techniques is that the control lattice is not directly related to the object.

Another class of freeform geometric modeling techniques relies on implicit geometric representations such as variational implicit surfaces [Cuno et al., 2005], spherical implicit surfaces [Alexe et al., 2004] and convolution surfaces [Bloomenthal and Shoemake, 1991]. An effective way of creating new shapes with implicit representations is using Boolean operations [Bajaj et al., 1997]. Another commonly used technique for implicit shape modeling is to extract the skeleton of a model and manipulate it to change the shape [Yoshizawa et al., 2003]. However, providing local modification capability is difficult since changing implicit functions is not very intuitive.

The spatial warping method has recently become popular due to its intuitive user interface and design operations. In this method, the user's gesture such as his/her hand's position and orientation is obtained with a mouse or a hand tracking device, and is utilized to define the space transformation matrix. Weights can be associated with the transformation matrix to control the gesture's influence area. Angelidis et al. [2006] developed a gesture based swept deformation method called the sweepers, where the transformation is divided into a series of small steps to avoid foldover problems with a lower bound of the required number of steps. Angelidis et al. [2004] also developed a gesture based deformation method capable of preserving the volume and avoiding self-intersections. Gain and Marais [2005] developed a warp sculpting method, which allows deformation to be initialed by the rigid body transformation of uniform scaling of volumetric tools. Joo et al. [2006] introduced a 3D warp brush method for interactive shape modeling in an immersive virtual reality environment. The deformed model is capable of adaptive refinement and efficient rendering with on-the-fly triangular strip generation.

However, the spatial warping method may generate a foldover of the ambient space and potential self-intersection of the embedded object, resulting a physically unrealistic, non-manifold object. Previous methods tried to decompose the transformation into a series of smaller transformations, and applied each of them to the result of the previous transformation to remove the foldover [Gain and Dodgson, 2001]. Choosing the right number of subdivisions is challenging because too few steps in the subdivisions will not prevent the fold-over and two many steps will jeopardize the interactivity of the system. Gain and Dodgson [2001] came up with a set of conditions for a self-intersection test to prevent fold-over from happening. However, an accurate test is often costly and unrealistic for real-time applications. Angelidis et al. [2006] derived a bound for the number of steps required for a fold-free deformation, but it has not been fully validated. Our method presented in this paper is the first to utlize the implicit shape representation based on the level-set method for modeling of spatial warping. The developed method guarantees the generation of a fold-free model.

2.2. Level-Set Method for Modeling of Freeform Geometry

The level-set method introduced by Osher and Sethian [1988] provides mathematical and numerical mechanisms for computing surface deformations as the time-varying iso-values of a function by solving a partial differential equation on the 3D grid. A set of numerical techniques has been provided by the level-set formulation that describes how to manipulate the distance values of each grid in a volume, so that the iso-surfaces of the function move in a prescribed manner. Museth et al. [2002] defined a collection of speed functions that produce a set of surface editing operators like blending, smoothing, sharpening, opening/closing and embossing. Bærentzen and Christensen [2002] developed a volume sculpting system by using the level-set method and introduced a scaling–window technique to define a speed function for local manipulations (e.g. smoothing, material addition/subtraction). Guo et al. [2004] applied the level-set method to model complicated point-set surfaces of arbitrary topology, allowing local surface editing and global scalar-field freeform deformation. They developed a variety of editing toolkits to directly manipulate the point-set surface through interactive sketching, smoothing, embossing, and global freeform deformations. Lawrence and Funkhouser [2004] developed a painting interface which allows the user to define the instantaneous surface velocity to deform the geometry using the level-set method. However, none of the previous studies has developed a spatial warping technique based on the level-set method.

3. THE SPATIAL WARPING METHOD

Spatial warping is a general freeform deformation approach that enables the use of a variety of virtual tools to interact with a CAD model. Unlike the "direct" modeling such as virtual sculpting [Leu et al., 2001, 2005], in spatial warping method, the user controls the position and orientation of a tool to generate a warp field that "indirectly" deforms the object that is inside the tool's influence zone. Once the new positions of all the vertices have been calculated, the tool's position and orientation are updated and ready for the user to continue shape modeling.

Our space-warping method works as follows: the user first selects a virtual tool from a tool library. Each tool has a limited region of influence in the 3D space

represented by the distance field. As the user moves the virtual tool around in the 3D space, a vector field is generated by the trajectory of the virtual tool. This vector filed defines the velocities for the grid points inside the influence zone of the tool. Then by solving the level-set differential equation that describes the shape deformation, the shape is deformed accordingly. A schematic of the freeform modeling system is shown in Fig. 1.

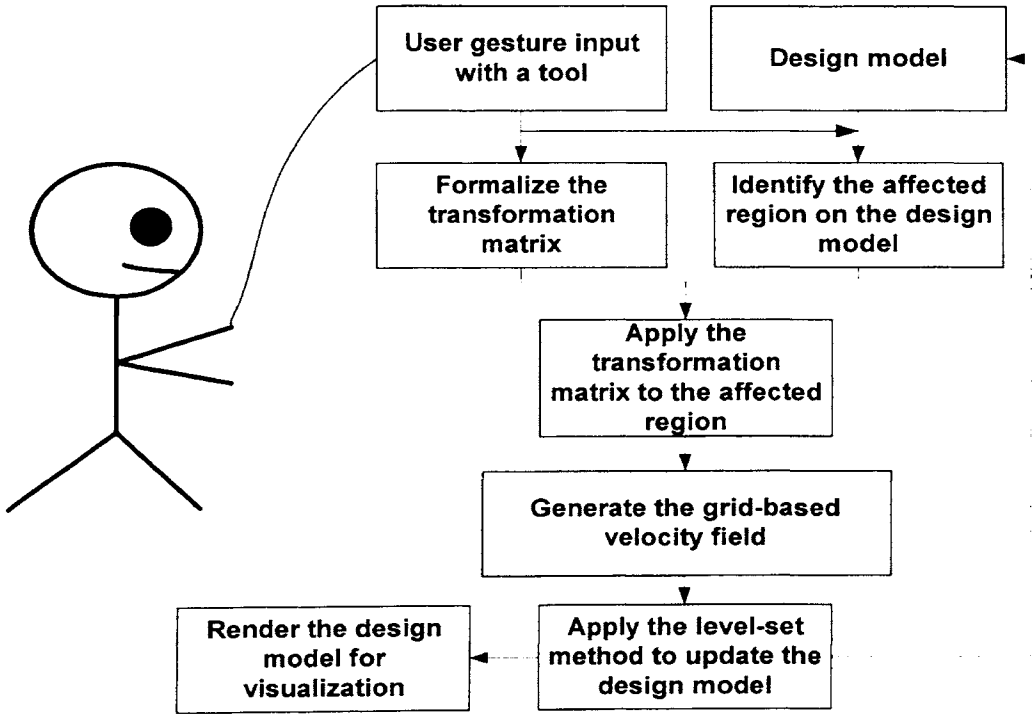


Figure 1. Schematic of the Freeform Modeling System

3.1. Input Data

The input of the modeling framework is a series of hand gestures (i.e., positions and orientations of user's hand), G_i (i=0,…,n), which can be obtained from a motion capture deivce. The gesture G_i at time t_i is defined by a local coordinate system with

origin O_i and three unit vectors u_i, v_i, w_i as shown in Fig. 2, where $u_i \times v_i = 0$, $v_i \times w_i = 0$ and $w_i \times u_i = 0$.

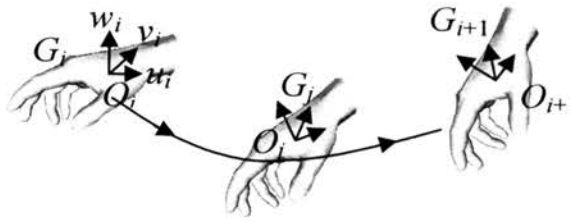


Figure 2. Hand Gesture Modeling by Interpolation

To produce a smooth space warping from the input gestures, a B-Spline interpolation is constructed to calculate the position and orientation of the gesture in between. The B-spline curve passing through (n+1) points, $O_i(x,y,z)$, i=0, … n, is defined as:

$$O_r(x,y,z) = \sum_{i=0}^{n} O_i(x,y,z) N_{i,k}(r) \tag{1}$$

where n+1 is the total number of sampled points from the user's hand input, r is the parameter, k-1 is the degree of the B-Spline curve and $N_{i,k}$ is the B-spline basis function where

$$N_{i,k}(r) = \frac{(r - t_i) N_{i,k-1}(r)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - r) N_{i+1,k-1}(r)}{t_{i+k} - t_{i+1}} \tag{2}$$

and

$$N_{i,1}(r) = \begin{cases} 1 & \text{if } t_i \le r \le t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

For a B-spline function, the parameter t_p is calculated as:

$$t_p = \begin{cases} 0 & if \quad p < k \\ p-k+1 & if \; k \le p \le n \\ n-k+2 & if \quad p > n \end{cases} \tag{4}$$

Similarly, the interpolated orientation (u_r, v_r, w_r) is calculated as

$$u_r = \sum_{i=0}^{n} u_i N_{i,k}(r) \tag{5}$$

$$v_r = \sum_{i=0}^{n} v_i N_{i,k}(r) \tag{6}$$

$$w_r = \sum_{i=0}^{n} w_i N_{i,k}(r) \tag{7}$$

The tangent vector of the B-spline curve at point $O_r(x, y, z)$ is calculated as

$$O'_r(x,y,z) = \sum_{i=0}^{n} O_i(x,y,z) N'_{i,k}(r) \tag{8}$$

3.2. The Influence Zone of a Tool

In the modeling method we propose, each virtual tool has a limited local region of space around the tool, defined by the distance field and a user defined region of influence (RI). The distance field $d_s(x)$ is a scalar field that is defined by the minimum distance between every point x in space and a given surface S. Because the tool shape in the modeling process is pre-defined, the tool's distance field can be preprocessed using the closest point transformation algorithm [Mauch, 2003]. The user can change the parameter R to adjust the tool's RI as shown in Fig. 3.

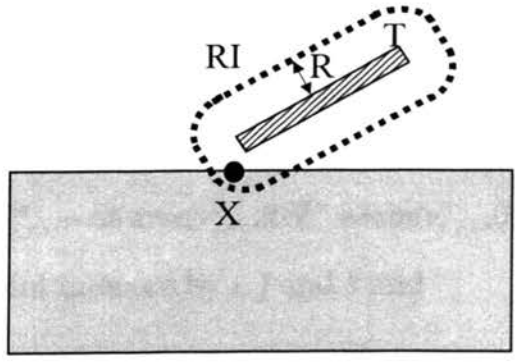


Figure 3. The Influence Zone of a Tool

The vertex of the workpiece inside the tool's RI is affected by the tools' movement. For example, in Fig. 3, the tool T is a rectangular block, the dotted line around the tool is the boundary of the tool's RI defined by distance R from the tool boundary. The vertex X on the boundary of the workpiece shown in Fig. 3 is inside the RI, thus it is transformed by the movement of the virtual tool T. Using the pre-calculated distance field $d_T(X)$ for the tool, it is simple to evaluate whether a point X is inside, outside or one the boundary of the tool's RI using the flowing relations:

$$d_T(X) - R \begin{cases} > 0 \Rightarrow \text{outside RI} \\ = 0 \Rightarrow \text{on boundary} \\ < 0 \Rightarrow \text{inside RI} \end{cases} \tag{9}$$

3.3. Shape Modeling Using the Level-Set Method

We utilize the level-set method to change the initial shape of the CAD model under the grid –based velocity field generated by the movement of the user's hand. In the level-set method, the geometry of an object is represented by an implicit distance field data. In this data, every grid point has a 3D coordinates as well as a signed Euler distance to the boundary of the object. The change of this distance field data is controlled by a velocity function ($\bar{v}$) in the level-set partial differential equation [Osher and Sethian, 1988]:

$$\frac{\partial F}{\partial t} = -\nabla F \cdot \bar{v} \tag{10}$$

where F(**x**,t) is the Euclidean distance function, **x** is the grid coordinates in Euclidean space R^3, $\bar{v}$ is the velocity of the boundary grid point, and ∇ is the gradient function

$$\nabla = i \cdot \frac{\partial}{\partial x} + j \cdot \frac{\partial}{\partial y} + k \cdot \frac{\partial}{\partial z} \tag{11}$$

where i, j and k are the unit vectors in R^3. To solve the level-set differential equation given in Eq. (10), an up-wind scheme [Osher and Sethian, 1988] is used with the first-order space approximation of the distance function given below:

$$F_{i,j,k}^{n+1} = F_{i,j,k}^{n} - \Delta t[\max(v_{i,j,k}^{n},0)\nabla^{+} + \min(v_{i,j,k}^{n},0)\nabla^{-}] \tag{12}$$

where $v_{i,j,k}$ is the speed at a point indexed by i, j and k and

$$\nabla^{+} = \begin{bmatrix} \max(D_{i,j,k}^{-x},0)^2 + \min(D_{i,j,k}^{+x},0)^2 + \\ \max(D_{i,j,k}^{-y},0)^2 + \min(D_{i,j,k}^{+y},0)^2 + \\ \max(D_{i,j,k}^{-z},0)^2 + \min(D_{i,j,k}^{+z},0)^2 \end{bmatrix}^{1/2} \tag{13}$$

$$\nabla^{-} = \begin{bmatrix} \max(D_{i,j,k}^{+x},0)^2 + \min(D_{i,j,k}^{-x},0)^2 + \\ \max(D_{i,j,k}^{+y},0)^2 + \min(D_{i,j,k}^{-y},0)^2 + \\ \max(D_{i,j,k}^{+z},0)^2 + \min(D_{i,j,k}^{-z},0)^2 \end{bmatrix}^{1/2} \tag{14}$$

where $D_{i,j,k}^{+x}$ is a shorthand notation of the forward difference operator $\frac{F_{i,j,k}(x+h,t) - F_{i,j,k}(x,t)}{h}$ and $D_{i,j,k}^{-x}$ is a shorthand notation of the backward difference operator $\frac{F_{i,j,k}(x,t) - F_{i,j,k}(x-h,t)}{h}$. The implementation of the level-set method is sped up using a narrow-band scheme developed by Osher and Sethian [1998]. The idea of this method is to update only the narrow-band of grid points which are close to the iso-surface, instead of the grid points in the entire region of concern. As a result, the number of points being computed is much smaller so that it is feasible to use a linked-list structure to keep track of them for real-time applications. By updating the distance values of the boundary grid points according to Eq. (10), the change of the 3D model's iso-surface can be tracked.

3.4. Grid-Based Velocity Field

The inputs of the level-set method are the model of the workpiece represented by the distance field data and external velocities on the grid points, called the grid-based velocity field (GVF). To generate the GVF from the trajectory of the user's hand, we first search for the workpiece's boundary vertices inside the tool's influence zone and calculate their trajectories caused by the tool movement. Then, for a grid point be on one or more of the grids (called the swept grid), intersected by one of these trajectories, we calculate the closest point on the trajectory and its tangent vector, which defines the direction of the velocity of the swept grid point.

Let a workpiece's boundary vertex $p_i(x_i, y_i, z_i)$ be inside the tool's RI as shown in Fig. 4. Its transformed point p_{i+1} can be calculated as

$$p_{i+1} = p_i + [\Delta u \cdot \Delta v \cdot \Delta w \cdot (p_i - O_i) + \Delta O] \tag{15}$$

where $\Delta u = u_{i+1} - u_i$, $\Delta v = v_{i+1} - v_i$, $\Delta w = w_{i+1} - w_i$ and $\Delta O = O_{i+1} - O_i$. The rest of points, $p_{i+2}, \ldots, p_n$, can also be calculated as above. Then, by utilizing Eqs. (1) - (7), a B-spline

function is formulated to interpolate the trajectory traversed through these points. This interpolated B-spline curve intersects many grids as shown in Fig. 4. Let g_j be a grid point on one or more of these grids. To calculate the velocity of g_j, we find its closest point p_j on the B-spline curve and calculate its tangent vector by Eq. (8). This is seen as a small dark arrow in Fig. 4. If the magnitude of the velocity is constant, it can be used to define an imprinting operation. If it is associated with a weighting function which varies with the distance from the grid point to the tool boundary, it can be used to define a deformation operation. If it is associated with the curvature of local geometry, it can be used to define a smoothing operation. These operations will be discussed in detail in Section 4.

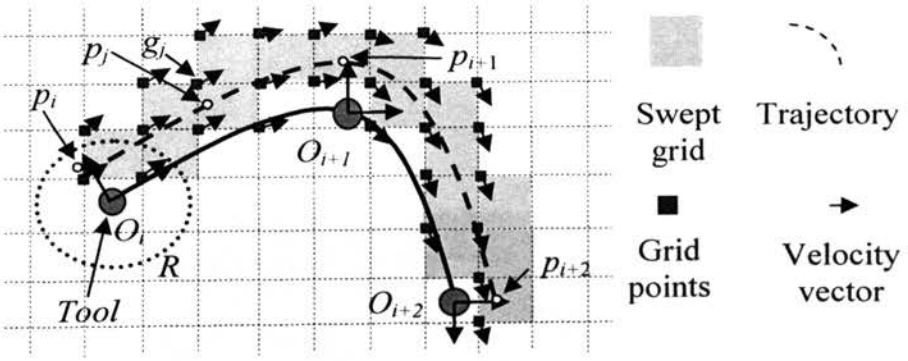


Figure 4. Generation of the Grid-Based Velocity Field

4. FREEFORM MODELING OPERATIONS

In this section, three freeform modeling operations, i.e. the imprinting operation, the deformation operation, and the smoothing operation, are developed. The modeling results are compared with other virtual sculpting methods to demonstrate the usefulness of the proposed method.

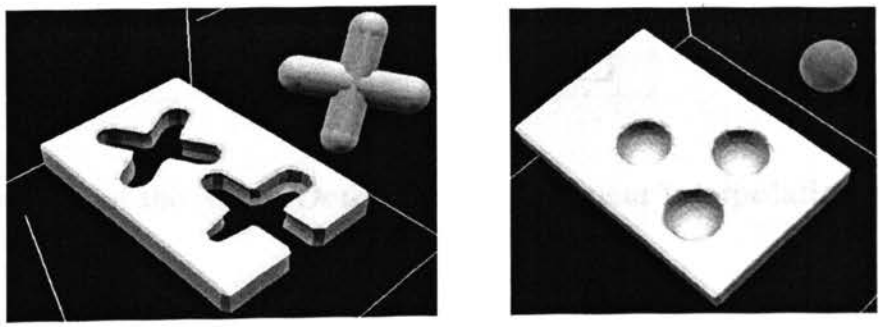
4.1. Imprint Operation

The imprinting operation works as follows: the user selects different types of tools from a tool library, and defines parameters to customize the tool shape. Then, the user

grabs the virtual tool using a 3D digital manipulating device such as a space mouse or the Phantom^{TM} haptic device, and applies the imprinting operation onto the initial workpiece model which is updated in real-time. In the imprinting operation, the movement of the virtual tool generates a grid-based velocity field along the path of the virtual tool and the solution of the level-set method changes the boundary of the workpiece. Only the velocity component in the normal direction of the workpiece boundary contributes to the change of this boundary, thus Eq. (10) can be written as

$$\frac{\partial F(x,t)}{\partial t} + k \cdot \| \nabla F(x,t) \| = 0 \tag{16}$$

where k is a user-defined constant to control the speed of the propagation. The gradient $\nabla F(x,t)$ is approximated by the up-wind finite difference scheme described in Sec. 3.3. Figure 5 is an example showing the imprinting operation by using a cross-shaped tool and a spherical tool to modify a rectangular plate.



(a) (b)

Figure 5. Example of Imprinting Operation. a) Using a Cross-Shaped Tool and (b) Using a Spherical Tool

4.2. Fold-Free Deformation Operation

As discussed in Sec. 3.4, the movement of a virtual tool generates a grid-based velocity field along the trajectory of the tool. To generate the effect of deformation, the

original grid-based velocity field is modified by a user-defined weight function as follows:

$$p_{i+1} = p_i + w(p_i) \cdot t(O_i) \tag{17}$$

where O_i is the position of the tool at time i, p_i is a vertex on the boundary of the workpiece inside the tool's region of influence, $w(p_i)$ is a user defined weight function to control the shape of the deformation, and $t(O_i)$ is the transformation matrix. Figure 6 is a 2D illustration example, where the tool moves from point O_i to O_{i+1}. The tool's influence zone is within the dotted ellipse at time i. The point p_i, which is a vertex of the workpiece surface inside the influence zone, is transformed to point p_{i+1} according to Eq. (17).

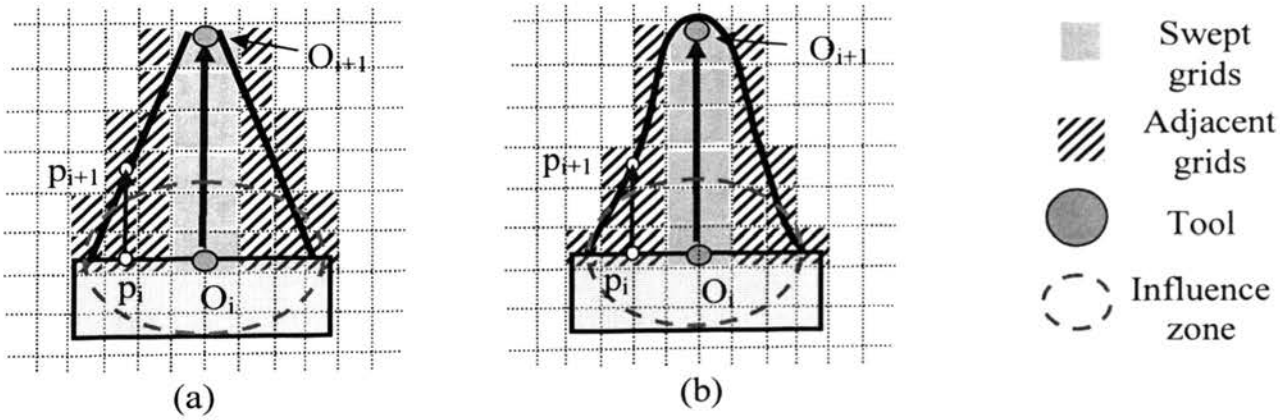


Figure 6. Example of the Shape Deformation. (a) Linear Interpolation and (b) Cubic Interpolation

The weight function can be defined as a linear interpolation or a cubic interpolation as follows:

$$w(x) = 1 - d(p) \tag{18}$$

$$w(x) = 1 - d^2(p)(3 - 2d(p)) \tag{19}$$

By using the above weight function, the top boundary of the workpiece can be deformed into different shapes as shown in Fig. 6. The grid-based velocity field is generated using the same procedure as given in Sec. 3.4.

As mentioned before, mesh-based spatial deformation method may generate fold-over of the ambient space and self-intersection of the object as shown in Fig. 7(a), where the upper boundary of the workpiece is deformed by the movement of tool from p to p' and intersected with the lower boundary of the shape. The deformed upper boundary is represented by the dotted lines.

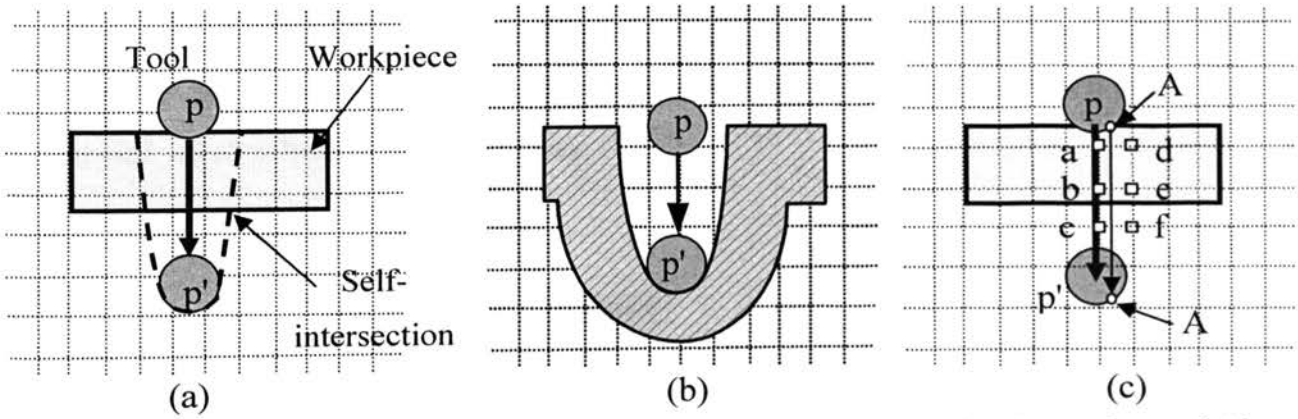


Figure 7. The Deformation Operation. (a) Self-Intersection, (b) the Deformed Shape Without Self-Intersection, and (c) Boundary Propagation by Defining the Velocity for the Boundary Grids

To solve the self-intersection problem and generate the deformed shape as shown in Fig. 7(b), we can calculate the grid-based velocity field not only according to the movement of the tool and the user-defined weight function, but also to the grid point's inside/outside information. We propose the following folder-free deformation algorithm consisting four steps:

- Step1: Identify the workpiece's boundary vertices inside the influence zone of the tool and their transformed points according to the input vector and a user defined

weight function. In Fig. 7(c), a boundary vertex, point A, is transformed to point A' under the input vector pp'.

- Step2: Identify the boundary grid points whose grids are intersected by the vector connecting a boundary vertex and its transformed point, and are adjacent to the iso-boundary. In Fig. 7(c), points *a* to *f* are the boundary grid points intersected by AA'.
- Step3: Calculate the velocities of the identified boundary grid points. For each boundary grid point, it is easy to calculate the surface normal using the central definite difference scheme. According to the level-set method, the boundary of the workpiece moves inwards if the velocity of the boundary grid point is negative and the boundary moves outwards if the velocity of the boundary grid point is positive. In order to move the boundary as desired to the target position as shown in Fig. 7(b), let the swept vector be T (=AA'), the surface normal of each boundary grid point be N, and the angle between T and N be α. We define the sign of the boundary grid point's speed (v) as a boundary as follows:

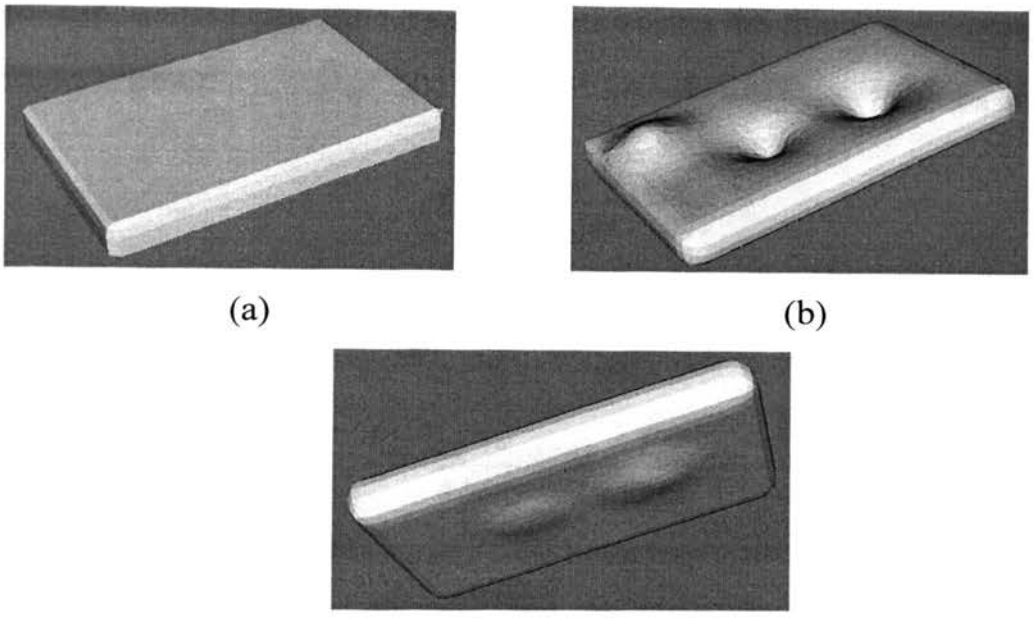
  If $(\alpha \in [90, \ 270])$

  $v<0$ (moving inward)

  Else $v>0$ (moving outward)

- Step4: Solve the level-set equation to update the boundary. With the boundary velocity calculated from the previous step, the numerical techniques given in Sec. 3.3 are utilized to update the boundary of the workpiece.

The above four steps are repeated until the boundary reaches the final location. After each step, the updated workpiece surface is generated by the marching cube algorithm. Because of using the level-set method for boundary propagation, the resultant surface model is guaranteed to be watertight without any self-intersection. An illustrative example is given in Fig. 8, where the user applies deformation operations onto a rectangular block.



(a) (b)

(c)

Figure 8. The Freeform Deformation Operation. (a) A Plate Model before Deformation (b) the Front Side of the Model after Deformation and (c) the Backside of the Model after Deformation

4.3. Smoothing Operation

In the smoothing operation, we assign the magnitude of the velocity at each grid point proportional to the curvature of the shape as follows:

$$\frac{\partial F(x,t)}{\partial t} - bH(x,t) \| \nabla F(x,t) \| = 0 \tag{20}$$

where b is a user-defined constant and $H(x,t)$ is the mean curvature of the boundary surface at point x, which is the average of the principal curvatures (κ_1 and κ_2), i.e.

$$H = (\kappa_1 + \kappa_2)/2 \tag{21}$$

For a surface in 3D space defined as $F(x,y,z)$, the mean curvature at a grid point is

$$H = \frac{(F_{yy} + F_{zz})F_x^2 + (F_{xx} + F_{zz})F_y^2 + (F_{xx} + F_{yy})F_z^2 - 2(F_xF_yF_{xy} + F_xF_zF_{xz} + F_yF_zF_{yz})}{2(F_x^2 + F_y^2 + F_z^2)^{3/2}} \tag{22}$$

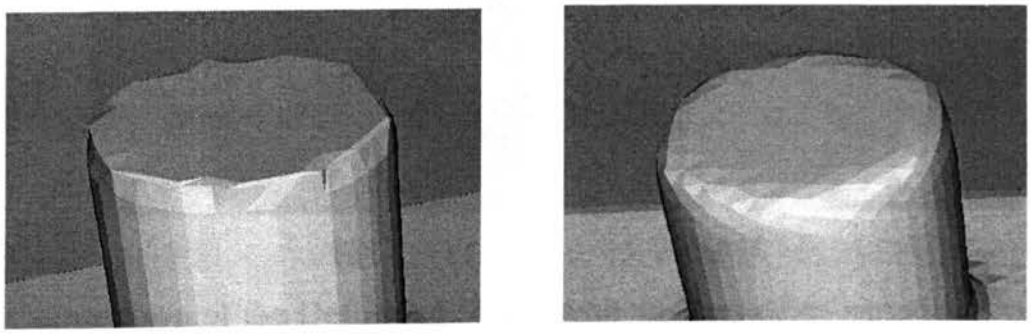
where the differential terms can be approximated using the first-order, central finite difference as follows:

$$F_x = \frac{F_{i+1,j,k} - F_{i-1,j,k}}{2\Delta x} \tag{23}$$

$$F_{xx} = \frac{F_{i+1,j,k} - 2F_{i,j,k} + F_{i-1,j,k}}{\Delta x^2} \tag{24}$$

$$F_{xy} = \frac{F_{i+1,j+1,k} - F_{i+1,j-1,k}}{4\Delta x \Delta y} + \frac{F_{i-1,j-1,k} - F_{i-1,j+1,k}}{4\Delta x \Delta y} \tag{25}$$

According to Eq. (20), the part of the boundary with a larger curvature moves faster along the surface normal direction than the part of the boundary with a smaller curvature. This movement results a smoothing operation as illustrated in Fig. 9, where the top of a cylindrical shape is smoothed by the developed smoothing operation.



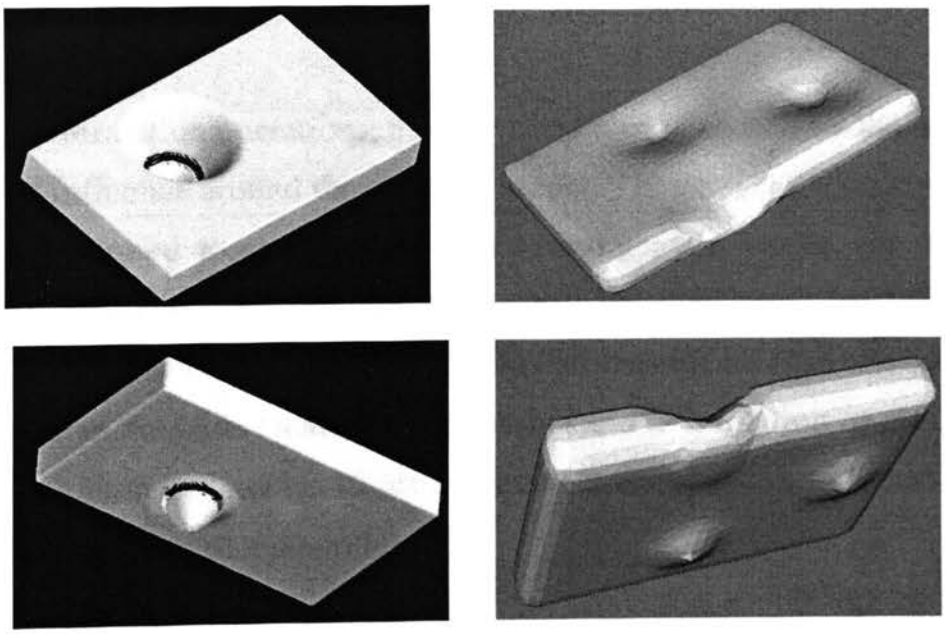
(a) (b)

Figure 9. Example of a Smoothing Operation on the Top of a Cylindrical Shape. (a) before Smoothing and (c) after Smoothing

4.4. Advantage of the Modeling Method

To demonstrate the advantage of our level-set based freeform modeling method with the same operation available from an existing commercial package, which is the FreeForm™ modeling system (v8.1) from SensAble Technology [2008], a thin

rectangular block is deformed to generate a dent area as shown in Fig. 10. In the FreeForm system, the deformed top surface intersects with the unreformed bottom surface and this self-intersection of the boundary of the rectangular block creates a non-manifold object with two separate geometric entities as seen in Fig. 10(a). In contrast, by using the level-set method, the entire shape is deformed without producing multiple parts, thus remaining a manifold, as seen in Fig. 10(b).



(a) (b)

Figure 10. Comparison of the Deformed Shape with the Same Deformation Operation. (a) by the FreeForm™ System and (b) by Our System

5. IMPLEMENTATION

Our freeform modeling system runs on a Microsoft Windows XP workstation equipped with a 1.6 GHz CPU and 1 GB RAM. The software is written in C++, and the graphics-rendering component is built upon OpenGL and GLUT libraries. The setup of the modeling system is shown in Fig. 11.

Figure 11. The Virtual Shape Modeling System Setup

To apply deformation operations, a pre-defined tool is chosen by the user to select a certain region of influence around the sculpted model. Then the workpiece within selected region is deformed according to the user's hand gesture inputs. The surface modification process can be stopped at any time once a satisfied result has been obtained. Figures 12(a) and (b) show two spheres joined together before and after the smoothing operation. A snowman model is created by smoothing and deformation on the two-sphere model and the result is shown in Fig. 12 (c). Figures 12(d) and (e) show a part of the snowman model before and after smoothing.

To evaluate the performance of the described method, a smoothing operation is performed on a shape. The number of grid points, the time of calculating distance values, and the time of updating the lists are given in Table 1. It can be seen from the table that about a 11.7Hz refresh rate can be achieved by updating 28,260 grid points in each iteration.

6. CONCLUSION

This paper presents the development of a spatial warping method using the implicit distance field data representation and the level-set method for shape modeling. The trajectory of the user's hand is interpolated and utilized to define a grid-based velocity field. The solution of the level-set method propagates the boundary of the workpiece with the external velocity field, resulting different freeform modeling

operations such as imprinting, deformation, smoothing, etc. The developed modeling operations are intuitive and easy to use for freeform modeling. Compared with the mesh-based spatial warping methods, the triangular meshes generated using the described spatial warping method are free of the self-intersection problem.

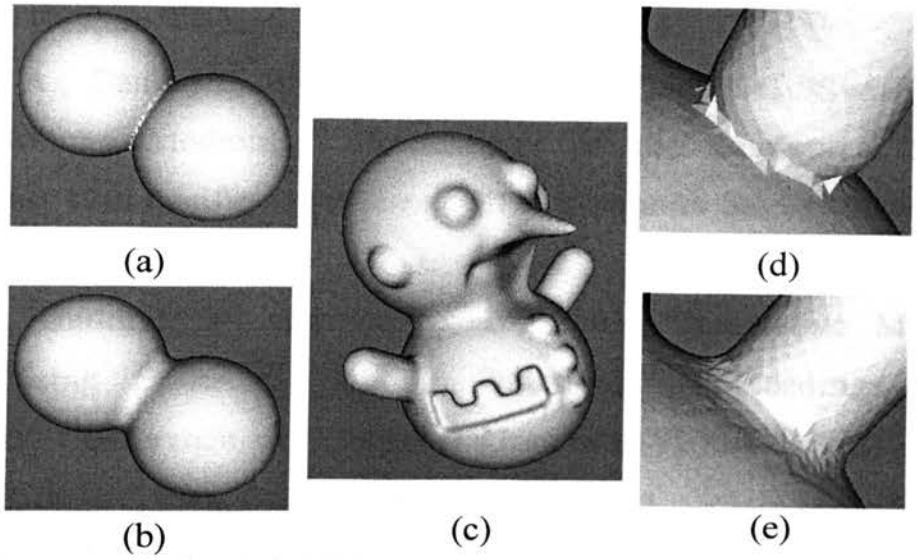


Figure 12. Modeling Example. (a) and (b) Two Joined Spheres and the Smoothed Shape, (c) the Snowman Model after Deformation and Smoothing, (d) & (e) Part of the Snowman Model before and after Smoothing

Table 1. Test Results of the Level-Set Method

No. of grid points	Time of calculating the distance values (s)	Time of updating the lists (s)	Total time (s)
202,592	0.4637	0.1631	0.6268
156,702	0.3675	0.0973	0.4648
149,942	0.3680	0.0902	0.4582
101,788	0.1754	0.0897	0.2651
28,260	0.0746	0.0108	0.0854
23,217	0.0638	0.0108	0.0746

ACKNOWLEDGMENTS

This research is supported by a National Science Foundation award (CCR-0310619) and by the Intelligent Systems Center at the Missouri University of Science and Technology.

REFERENCES

1. Angelidis, A., Wyvill, G., and Cani, M., 2006, "Sweepers: Swept Deformation Defined by Gesture," Graph. Models, 68(1), pp. 2-14.
2. Angelidis, A., Cani, M., Wyvill, G., and King, S., 2004, "Swirling-Sweepers: Constant-Volume Modeling," Proceedings of the Computer Graphics and Applications, 12th Pacific Conference, IEEE Computer Society, Washington, DC, pp. 10-15.
3. Alexe, A., Gaildrat, V., and Barthe, L., 2004, "Interactive Modeling from Sketches using Spherical Implicit Functions," Proceedings of the 3rd international Conference on Computer Graphics, Virtual Reality, Visualization and interaction in Africa, Stellenbosch, South Africa.
4. Bærentzen, J. A., and Christensen, N. J., 2002, "Volume Sculpting Using the Level-Set Method," Proceedings of Shape Modeling International'02, pp. 175-182.
5. Bajaj, C., Blinn, J., Bloomenthal, J., Cani-Gascuel, M.P., Rockwood, A., Wyvill, B., and Wyvill, G., 1997, Introduction to Implicit Surfaces, Morgan-Kaufmann Publisher.
6. Bill, J. R., and Lodha, S. K., 1995, "Sculpting Polygonal Models using Virtual Tools," Proceedings of Graphics Interface, Quebec, Canada, pp. 272-279.
7. Bloomenthal, J., and Shoemake, K., 1991, "Convolution Surfaces," Computer Graphics, 25(4), pp. 251--256.
8. Coquillart, S., 1990, "Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling," Computer Graphics, 24(4), pp. 187-196.
9. Cuno, A., Esperana, C., Roma, P., and Farias, R., 2005, "3D Free-Form Modeling with Variational Surfaces," Journal of Winter School of Computer Graphics, 3(1-3), pp. 111-122.

10. FreeForm Modeling and Modeling Plus Systems, SensAble Technology, 2008, http://www.sensable.com/products-freeform-systems.htm.
11. Gain, J. E. and Dodgson, N. A., 2001, "Preventing Self-Intersection under Free-Form Deformation," IEEE Transactions on Visualization and Computer Graphics, 7(4), pp. 289-298.
12. Gain, J., and Marais, P., 2005, "Warp Sculpting," IEEE Transactions on Visualization and Computer Graphics, 11(2), pp. 217-227.
13. Guo, X., Hua, J., and Qin, H., 2004, "Point Set Surface Editing Techniques Based On Level-Sets," Proceedings of the Computer Graphics International, Geneva, Switzerland pp. 52-59.
14. Hsu, W. M., Hughes, J. F., and Kaufman, H., 1992, "Direct Manipulation of Free-Form Deformation," Computer Graphics, 26(2), pp. 177-184.
15. Joo K. Y., Renzulli, P., Kreylos, O., Hamann, B., Monno, G., and Staadt, O. G., 2006, "3D Warp Brush Modeling," Computers & Graphics, 30(4), pp. 610-618.
16. Lamousin, H. J., and Waggenspack, W. N., 1994, "NURBS-Based Free-Form Deformations," IEEE Computer Graphics & Applications, pp 59-65.
17. Lawrence, J. and Funkhouser, T., 2004, "A Painting Interface for Interactive Surface Deformations," Graphical Models, 66(6), pp. 418-438.
18. LeBlanc, A., Kalra, P., Magnenat-Thalmann, N., and Thalmann, D., 1991, "Sculpting with the 'Ball and Mouse' Metaphor," Proceedings of graphics Interface, Calgary, Alberta, Canada, pp. 152-159.
19. Leu, M.C., Maiteh, B. Y., Blackmore, D., and Fu, L., 2001, "Creation of Freeform Solid Models in Virtual Reality," Annals of CIRP, 50(1), pp. 73-76.
20. Leu, M.C., Peng, X., and Zhang, W., 2005, "Surface Reconstruction for Interactive Modeling of Freeform Solids by Virtual Sculpting," Annals of CIRP, 54(1), pp. 131-134.
21. Mauch, S., 2003, "Efficient Algorithms for Solving Static Hamilton-Jacobi Equations," PhD dissertation, California Institute of Technology, Pasadena, CA.
22. Museth, K., Breen, D. E., Whitaker, T., and Barr, A. H., 2002, "Level-Set Surface Editing Operators," ACM Transactions on Graphics, 21(3), pp. 330-338.

23. Osher, S., and Sethian, J., 1988, "Fronts Propagating with Curvature-Dependent Speed: Algorithms based on Hamilton-Jacobi Formulations," Journal of Computational Physics, 79, 12–49.
24. Parent, R., 1977, "A System for Sculpting 3-D Data," Computer Graphics, 11(2), pp. 138-147.
25. Sederberg, T.W., and Parry, S.R., 1986, "Freeform Deformation of Solid Geometric Models," Computer Graphics, 20(4), pp. 151-160.
26. Yoshizawa, S., Belyaev, A. G., and Seidel, H. 2003. "Free-Form Skeleton-Driven Mesh Deformations," Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications, Seattle, Washington, pp. 247-253.

APPENDIX

PSEUDO CODES OF THE CONTOUR GENERATION AND CONTOUR COMBINATION ALGORITHMS

The pseudo code of the contour generation algorithm is given below:

Procedure: Search contour points from the three-column table

```
BEGIN
//variables
variable nContour              //the number of contours
variable ContourArray[ ]       //the array used to store the point indexes of each contour
variable MiddleColumn[ ]       //the array used to store the point indexes in the middle column
variable LeftColumn[ ]         //the array used to store the point indexes in the left column
variable RightColumn[ ]        //the array used to store the point indexes in the right column
variable Point[ ]        //the array used to store the coordinates of points and the traverse
information
variable Index           //the index of the point in the MiddleColumn[ ] and Point[ ]
variable LeftIndex             //the index of the point in the LeftColumn[ ]
variable RightIndex            //the index of the point in the RightColumn[ ]

Index = 0
nContour = 0
    REPEAT
        REPEAT
            ContourArray[nContour].addPoint (Point[Index].coordinates)
            LeftIndex  = LeftColumn[Index].getPointIndex
            RightIndex = RightColumn[Index].getPointIndex
            IF Point[LeftIndex].traversed = false
                    Index = LeftIndex
            ELSE
                    Index = RightIndex
```

END IF

Point[Index].traversed = true

UNTIL

Point[LeftIndex].traversed = true && **Point[RightIndex].traversed** = true

nContour++

Index = the index of next unsearched point in the middle column

UNTIL all the points in the middle column are traversed

END

The pseudo code for Step 1 of the contour combination algorithm is given below where A and B represent contour A_i and B_j, and a_i represents the point in contour A with index i.

Procedure: Search starting points from contour A and associated points from contour B

A: list of points in contour A

B: list of points in contour B

C: list of points in the combined contour

f: the first associated point in contour B

f': candidates for the first associated point in contour B

l: the last associated point in contour B

l': candidates for the last associated point in contour B

DexelY: dexel data in y direction

FOR (i=A→begin(); i ≠ A→end(); i++)

IF ($\mathrm{INT}[(a_i \rightarrow [x])/\Delta x] < \mathrm{INT}[(a_{i+1} \rightarrow [x])/\Delta x]$

THEN

templist1=DexelY [INT[(a_i→[x])/ Δx]+ Δx] [a_i→[z]];

templist2=DexelY [INT[(a_{i+1}→[x])/ Δx]] [a_{i+1}→[z]];

FOR every point p in templist1 and every point q in templist2

IF ($p \rightarrow [y] \in [a_i \rightarrow [y] - \Delta y, a_i \rightarrow [y] + \Delta y]$

THEN save p to f'

ENDIF

IF ($q \rightarrow [y] \in [a_{i+1} \rightarrow [y] - \Delta y, a_{i+1} \rightarrow [y] + \Delta y]$

THEN save q to l'

ENDIF

ENDFOR

IF ($a_i \rightarrow$[y] $\neq$ $a_{i+1} \rightarrow$[y])

THEN

IF (f'$\rightarrow$number() =1) **AND** (l'$\rightarrow$number() =1)

THEN

f=f'; l=l';

return;

ELSE

break;

ENDIF

ELSE //($a_i \rightarrow$[y] = $a_{i+1} \rightarrow$[y])

IF (f'$\rightarrow$number() >2) **OR** (l'$\rightarrow$number() >2)

THEN

break;

ELSE

find f and l according to one of the cases CA1, CA2, CA3 and CA4;

return;

ENDIF

ENDIF

ENDIF

ENDFOR

Search contour B to find the index of point f

Renumber points in contour B by indexing point f as b_1 without affecting the point sequence

Search the renumbered contour B to index point l as b_l

Renumber points in contour A by indexing point a_i as a_1 without affecting the point sequence

Add points a_1, b_1,..., b_l, a_2 to C

The pseudo code for Step 2 of the contour combination algorithm is given below.

Procedure: Search the rest of pairs from A and their associated points from B

l: index of the last associated point for the previous pair of points in contour A

r: point index in contour A

s: point index in contour B

o: index of the last associated point in contour B

r=2;

s=l+1;

REPEAT

IF (INT[$(a_r \rightarrow [x])/\Delta x$] ≠ INT[$(a_{r+1} \rightarrow [x])/\Delta x$])

THEN //search for b_o

FOR (t=s; t ≠ B→end(); t++)

IF ($a_{r+1} \rightarrow [y] \in [\min(b_t \rightarrow [y], b_{t+1} \rightarrow [y]),\ \max(b_t \rightarrow [y], b_{t+1} \rightarrow [y])]$) **AND**

(($b_t \rightarrow [x] = \mathrm{INT}(a_{r+1} \rightarrow [x]/\Delta x)$) **OR**

($b_t \rightarrow [x] = \mathrm{INT}(a_{r+1} \rightarrow [x]/\Delta x) + \Delta x$))

THEN

o=t; break;

END IF

END FOR

ELSE

IF ($a_r \rightarrow [y] \in [\min(b_{s-1} \rightarrow [y], b_s \rightarrow [y]),\ \max(b_{s-1} \rightarrow [y], b_s \rightarrow [y])]$) **AND**

(($\mathrm{INT}(b_s \rightarrow [x]/\Delta x) = \mathrm{INT}(a_r \rightarrow [x]/\Delta x)$ **OR**

($\mathrm{INT}(b_s \rightarrow [x]/\Delta x) = \mathrm{INT}(a_r \rightarrow [x]/\Delta x) + \Delta x$))

THEN

FOR (t=s; t ≠ B→end(); t++)

IF ($a_{r+1} \rightarrow [y] \in [\min(b_t \rightarrow [y], b_{t+1} \rightarrow [y]),\ \max(b_t \rightarrow [y], b_{t+1} \rightarrow [y])]$)

THEN o=t;

END IF

END FOR

ELSE

Add point a_r to the end of C;

r++;

break;

END IF

END IF

Add points a_r, b_s,…,b_o, a_{r+1} to the end of C;

s=o+1; r++;

UNTIL o=B→end() **AND** r=A→end()

VITA

Weihan Zhang was born in Beijing, China on February 1, 1978. He received his primary and secondary education in Beijing, China. He obtained his Bachelor of Science degree and Master of Science degree both in Mechanical Engineering from Tsinghua University, Beijing, China in June 2000 and June 2003, respectively. He went on to enroll in the Ph.D. program at the Department of Mechanical and Aerospace Engineering at the Missouri University of Science and Technology (formerly University of Missouri-Rolla) in USA. His research interests are in the field of geometric modeling, virtual reality, computer graphics and haptics, with special interests in freeform geometric modeling. He graduated with Doctor of Philosophy degree in Mechanical Engineering in December 2008.